

*О.Г. Трофименко,
Ю.В. Прокоп,
Н.І. Логінова,
О.В. Задерейко*



Алгоритмізація та програмування

ББК 32.973
УДК 004.43:004:421(076)

Р е ц е н з е н т и:

Положаєнко С.А. – доктор технічних наук, професор,
завідувач кафедри комп'ютеризованих систем управління
Одеського національного політехнічного університету

Кобозєва А.А. – доктор технічних наук, професор, завідувач кафедри
інформатики та управління захистом інформаційних систем
Одеського національного політехнічного університету

Рекомендовано Вченою Радою
Національного університету «Одеська юридична академія»,
протокол № 4 від 27 грудня 2018 р.

Трофименко О.Г., Прокоп Ю.В., Логінова Н.І., Задерейко О.В.
C++. Алгоритмізація та програмування : підручник.
2-ге вид. перероб. і доповн. Одеса: Фенікс, 2019. 477 с.

Підручник містить опис мови програмування C++ у поєднанні із засобами візуального програмування та об'єктно-орієнтованого підходу Visual C++. Розглянуто засоби програмування як базових алгоритмів, так і опрацювання структурованих типів, робота з вказівниками, засоби динамічного керування пам'яттю тощо. Кожна з розглянутих тем супроводжується значною кількістю прикладів програм.

Призначено для студентів технічних вузів і може бути корисною для кола читачів, які бажають навчатися програмуванню мовою C++.

ISBN 978-966-928-402-0

© О. Г. Трофименко,
Ю. В. Прокоп,
Н. І. Логінова,
О. В. Задерейко, 2019

ЗМІСТ

Передмова	5
Розділ 1 Основи алгоритмізації	8
1.1 Етапи розв'язування обчислювальних задач на комп'ютері.....	8
1.2 Алгоритм, його властивості й засоби описування. Поняття програми	8
1.3 Графічне подання у вигляді схеми алгоритму (блок-схеми).....	9
1.4 Види базових алгоритмів.....	11
1.5 Приклади базових алгоритмічних структур.....	12
Розділ 2 Знайомство з C++. Способи введення-виведення даних	15
2.1 Технологія створювання інтерфейсних програм у Visual C++	15
2.2 Введення-виведення даних.....	35
2.3 Робота в онлайн-компіляторах мовою C++	41
Розділ 3 Елементи мови C++	47
3.1 Історія C++	47
3.2 Алфавіт мови C++	48
3.3 Типи даних	50
3.4 Константи	54
3.5 Правила записування арифметичних виразів	55
3.6 Функції C++ генерування випадкових чисел	62
Розділ 4 Програмування базових алгоритмів.....	65
4.1 Програмування лінійних алгоритмів.....	65
4.2 Програмування розгалужених алгоритмів	71
4.3 Програмування циклічних алгоритмів.....	102
Розділ 5 Масиви в C++	149
5.1 Поняття масиву	149
5.2 Одновимірні масиви	149
5.3 Двовимірні масиви.....	175
Розділ 6 Вказівники. Динамічна пам'ять	206
6.1 Вказівники.....	206
6.2 Вказівники на одновимірні масиви.....	208
6.3 Арифметика вказівників	209
6.4 Вказівники на багатовимірні масиви.....	212
6.5 Динамічна пам'ять	216
6.6 Динамічні одновимірні масиви.....	218
6.7 Динамічні двовимірні масиви (матриці)	229
Розділ 7 Символи і рядки.....	240
7.1 Символьний тип даних.....	240
7.2 Рядки	246
7.3 Розширені символьні типи	299

Розділ 8 Функції	305
8.1 Правила організації функцій.....	305
8.2 Способи передавання параметрів до функцій.....	311
8.3 Параметри зі значеннями за промовчанням.....	319
8.4 Функції зі змінною кількістю параметрів.....	320
8.5 Рекурсивні функції.....	322
8.6 Перевантаження функцій	332
Розділ 9 Багатофайлові програми.....	337
9.1 Міжфайлова взаємодія	337
9.2 Заголовні файли.....	339
9.3 Директиви препроцесора.....	341
9.4 Область дії та простір імен.....	344
9.5 Бібліотеки функцій	349
9.6 Створення проектів з декількома формами.....	359
Розділ 10 Типи користувача	362
10.1 Перейменовування типів (typedef).....	362
10.2 Об'єднання (union).....	363
10.3 Структури (struct).....	364
10.4 Перерахування (enum).....	376
Розділ 11 Типи опрацювання дати і часу.....	380
11.1 Тип «дата-час» у C++	380
11.2 Дата-час у Visual C++.....	388
Розділ 12 Файли	402
12.1 Загальні відомості про файли	402
12.2 Текстові файли	403
12.3 Бінарні файли.....	432
Бібліографічний список	456
Додаток А Таблиці кодів ASCII.....	458
Додаток Б Операції мови C++.....	462
Додаток В Функції стандартної бібліотеки C++.	
Вміст заголовних файлів.....	464
Предметний покажчик	464

ПЕРЕДМОВА

Універсальною найпоширенішою мовою програмування упродовж кількох останніх десятиліть є мова C++, на підставі якої «виросло» багато сучасних мов програмування. Цьому сприяли такі її властивості як: лаконічність, потужність, гнучкість, мобільність, можливість доступу до всіх функціональних засобів операційної системи. C++ є ультрашвидкою кросплатформовою мовою загального призначення. Мовою C++ написано багато програм – від ігор, офісних застосунків, серверних частин сайтів (back-end), відеоплеєрів і до операційних систем. Кросплатформовість C++ є вагомою перевагою цієї мови, позаяк існують численні компілятори та інтерактивні середовища розробки (IDE), наприклад, , які мають версії для Windows, MacOS і Linux. До речі, Visual Studio – популярне інтегроване середовище розробки програмного забезпечення на різних платформах і для різних операційних систем з можливістю одночасної роботи над кодом декількох розробників (у версії Visual Studio 2019 – до 35 розробників).

Щоб не звужувати вибір читача щодо технічних засобів, а зосередити його на можливостях та особливостях мови C++, підручник подано як ґрунтовний і водночас доволі легкий для сприйняття матеріалу стосовно вивчення класичних методів програмування. Видання призначене як для початківців, які прагнуть навчатися програмувати мовою C++, так і для тих, хто має базові знання з програмування і прагне їх поглибити та освоїти більшість методів і прийомів, які має «на озброєнні» програміст. У підручнику містяться не лише тривіальні алгоритми, а й задачі з опрацювання структурованих типів даних, робота з вказівниками, засоби динамічного керування пам'яттю тощо. Значну увагу приділено опрацюванню даних у файлах.

Підручник може використовуватись як джерело прикладів, вправ та програм для розглядання. Автори намагались показати специфіку основних засобів та можливостей потужної й гнучкої мови C++ на значній кількості прикладів працездатних програм і програмних фрагментів, які можуть бути використані у різних версіях C++. Для реалізації прикладів програм використовувались:

- середовище розроблення Microsoft Visual Studio (Visual C++);
- <http://cpp.sh> – онлайн компілятор мовою C++;
- <https://ideone.com> – онлайн-компілятор різними мовами програмування;
- <http://codepad.org> – онлайн компілятор мовами C, C++ та ін.

Автори сподіваються, що численна кількість прикладів програм спонукатиме читача як реалізувати їх на комп'ютері, щоби перевірити їхню працездатність, так і змінити та вдосконалити ці програми.

Підручник складається з дванадцяти розділів.

На початку, у розд. 1 описані етапи розв'язування обчислювальних задач на комп'ютері та різновиди складання алгоритмів.

У розд. 2 описані інструментальні засоби інтерактивного середовища Microsoft Visual Studio (Visual C++) – однієї із найпотужніших систем, яка до-

звояє на найсучаснішому рівні створювати як окремі прикладні програми, так і розгалужені комплекси, призначені для роботи в корпоративних мережах і в інтернеті. Середовище Visual Studio поєднує засоби мови програмування C++ та компонентно-орієнтований підхід до створення програм. Поєднання простоти освоєння візуального середовища та підтримки широкого спектра технологій робить Visual Studio універсальним інструментом створення програмних проєктів якого завгодно рівня складності. Навіть початкові програмісти зможуть швидко створювати програмні проєкти у Visual C++, які матимуть професійний віконний інтерфейс найрізноманітнішої спрямованості, від суто обчислювальних та логічних, до графічних та мультимедійних. Поряд з цим, значну увагу в цьому розділі приділено створюванню консольних програм, оскільки їхній програмний код є універсальним і майже не відрізняється від програм інших засобів розроблення програм мовою C++.

Інформацію стосовно основних елементів мови C++, тобто щодо типів даних, правил оголошування змінних та інших об'єктів, записування арифметичних виразів, наведено у розд. 3. Цей розділ є надто важливим, позаяк для створювання програм треба знати синтаксис мови, тобто правила записування команд та використання лексичних одиниць мови.

У розд. 4 приділено увагу організації трьох основних видів обчислювальних процесів (лінійних, розгалужених і циклічних) та застосуванню різного роду операторів для їхньої програмної реалізації. Специфіку їхнього програмування ілюструє значна кількість прикладів програм та програмних фрагментів.

Широкого застосовування для зберігання та опрацювання однорідної інформації, наприклад, таблиць, векторів, рядків, матриць та багато ін., мають масиви, організації та опрацюванню яких присвячено розд. 5.

У розд. 6 йдеться про переваги та особливості організації динамічної пам'яті для структур даних, які змінюють розміри, з метою ефективного використання оперативної пам'яті. Така можливість пов'язана з наявністю та широким використанням у C++ особливого типу даних – вказівників.

Оскільки на практиці, крім чисел, доволі часто доводиться працювати із символьною інформацією, цій темі присвячений окремий розділ. Так, у розд. 7 докладно, з наведенням значної кількості наочних прикладів розглянуто особливості опрацювання різних видів організації символьної інформації: і як символьних масивів певної довжини, і за допомогою вказівників на початок символьного масиву, і як рядки.

Незамінну роль відіграють вказівники при передаванні параметрів до функції. У C++ підтримуються функції як окремі логічні одиниці (програмні конструкції) для виконання конкретних завдань. Наприклад, процес розроблення програмного забезпечення припускає поділ складного завдання на декілька більш простих завдань, кожне з яких розв'язується в окремій функції. Висвітленню специфіки організації функцій присвячено розд. 8.

Доволі часто функції організовують в окремі файли, особливості такого процесу описано у розд. 9.

Значну увагу у розд. 10 приділено складовому типу даних – структурі, яка, на відміну від масивів, може поєднувати дані різних типів. За допомогою

структур зручно зберігати списки даних, що складаються з полів різних типів. Окрім того, у цьому розділі вивчаються можливості створення типів користувачів на базі стандартних типів, особливості організації й застосування об'єднань, перераховного типу та множин.

У розд. 11 розглянуто спеціальні засоби C++ та Visual Studio .NET для опрацювання даних типу «дата-час». Проілюстровано значну кількість функцій та методів опрацювання такого роду даних.

Значну увагу в підручнику приділено специфіці організації та опрацювання файлів у C++, позаяк на практиці часто виникає потреба, щоб усі опрацьовувані дані зберігалися в окремих файлах. Так, у розд. 12 описано й продемонстровано на низці наочних прикладів використання потужних і гнучких можливостей C++ для різних підходів щодо організації файлів. Зазначено, що C++ припускає роботу з текстовими й бінарними файлами і має принципово різні способи їх опрацювання. У розділі висвітлюється специфіка роботи з файлами як з потоками у стилі C++. Окрім того, описано можливості роботи з файлами у Visual Studio за допомогою бібліотечних компонентів.

Щоб забезпечити ефективне сприйняття матеріалу та формування практичних навиків, наприкінці кожного розділу розміщені питання та завдання для самоконтролю засвоєння знань.

У додатки винесено таблиці ASCII-кодів, перелік операцій мови C++ у порядку зменшення пріоритету та перелік функцій стандартної бібліотеки C++. До предметного покажчика, який призначений полегшити користування підручником, включено основні терміни і поняття, що зустрічаються у цьому виданні.

Автори сподіваються, що запропонований до Вашої уваги підручник стане корисним при вивченні мови C++, і будуть щиро вдячні за доброзичливу та конструктивну критику й відгуки.

E-mail: egt@ukr.net.

РОЗДІЛ 1

ОСНОВИ АЛГОРИТМІЗАЦІЇ

1.1 Етапи розв'язування обчислювальних задач на комп'ютері

Хоча існують і використовуються сотні тисяч різноманітних прикладних програм, користувачі самі мають можливість розробляти власні програми для розв'язування тих чи інших задач на комп'ютері. Процес розроблення програми може бути подано як послідовність таких кроків:

1) *постановка задачі* – окреслення вимог щодо програми (наприклад, визначення початкових даних, їхнього формату, параметрів введення). Результатом цього етапу розроблення є докладний словесний чи математичний опис алгоритму;

2) *побудова математичної моделі задачі* – опис задачі за допомогою математичних формул, визначення переліку початкових даних та шуканих результатів, вихідні умови, точність обчислень;

3) *вибір методу розв'язування*, позаяк одну й ту саму задачу може бути розв'язано за допомогою різних методів. Вибір методу має визначатися багатьма чинниками, основними з яких є точність результатів, час розв'язування, обсяг займаної оперативної пам'яті. У кожному конкретному випадку за критерій вибору методу беруть один зі згаданих критеріїв;

4) *розроблення схеми алгоритму розв'язування задачі*, тобто оптимальної логічної послідовності дій з урахуванням вибраного методу розв'язування для здобуття певних результатів;

5) *написання та введення розробленої програми* алгоритмічною мовою програмування до комп'ютера;

6) *налагодження програми*, тобто пошук і виправлення можливих синтаксичних та алгоритмічних помилок у програмі;

7) *тестування* – перевірка правильності роботи програми.

Ця послідовність є притаманна для розв'язування якої завгодно задачі у програмний спосіб. Однак при підготовці задачі кожен етап може мати більш чи менш виражений характер.

1.2 Алгоритм, його властивості й засоби описування. Поняття програми

Алгоритм (algorithm) – це система формальних правил, які чітко й однозначно окреслюють послідовність дій обчислювального процесу від початкових даних до шуканого результату. Алгоритм визначає певні правила перетворення інформації, тобто він зазначає певну послідовність операцій опрацювання даних, щоб здобути розв'язок задачі.

Походження слова алгоритм: понад тисячу років тому у Багдаді жив Абу Джафар Мухамад ібн Муса аль-Хорезмі, який у своїй праці «Трактат аль-Хорезмі про арифметичне мистецтво індусів» з арифметики та алгебри сформулював правила і окреслив послідовність дій при додаванні та множенні чисел. При переведенні латиною ім'я автора перетворили в Algorithmi, а з часом методи розв'язування задач стали називати алгоритмами.

Основні властивості алгоритмів:

детермінованість (визначеність) – однозначність результату обчислювального процесу за заданих початкових даних;

дискретність – поділ обчислювального процесу на окремі елементарні кроки, виконувані послідовно;

ефективність – простота та швидкість розв'язання задачі за мінімальних потрібних засобів;

результативність – забезпечення здобуття розв'язку через певну кінцеву кількість кроків;

масовість – забезпечення розв'язання якої завгодно задачі з класу однотипних.

Задавати алгоритми можна у різні способи:

- ✓ словесний опис послідовності дій;
- ✓ аналітичний опис у вигляді формул;
- ✓ графічне подання у вигляді схеми алгоритму (блок-схеми);
- ✓ записування алгоритмічною мовою програмування.

1.3 Графічне подання у вигляді схеми алгоритму (блок-схеми)

Найчастіше алгоритми обчислювальних процесів подаються у вигляді блок-схем (графічний спосіб задавання алгоритмів), де кожний крок алгоритму подано спеціальним блоком, який показує виконувану дію.

Зазвичай графічному опису передують побудова математичної моделі – математичний опис алгоритму. Такий опис полягає у формалізованому (із застосуванням математичних символів) поданні всіх розглядуваних залежностей і методів обчислення значень вихідних даних на підставі вхідних.

Схема алгоритму (блок-схема) – графічне зображення його структури, в якому кожен етап процесу опрацювання даних подається у вигляді різних геометричних фігур (блоків).

Призначення блоків у схемі алгоритму впливає з їхніх назв. Форма блока зумовлює спосіб дій, а записи всередині – деталі (параметри) відповідного етапу (табл. 1.3). Біля блока можуть бути розміщені певні коментарі.

Блоки поєднуються поміж собою лініями потоку, які для кожного етапу визначають напрямок дій. Усім блокам присвоюють порядкові номери, які проставляються у розриві лінії контуру у лівій частині верхньої сторони зображення блока. Лінії потоку проводять паралельно до ліній зовнішньої рамки схеми. Напрямок ліній потоку зверху вниз і зліва направо прийнято за основний, і, як-

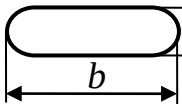
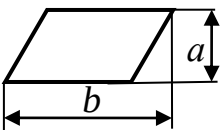
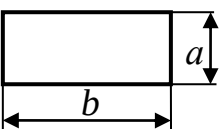
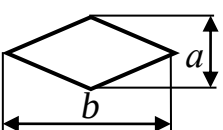
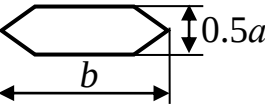
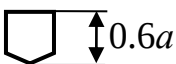
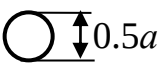
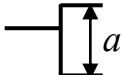
що лінії не мають зламів, стрілками їх можна не позначати. В інших випадках їхній напрямок обов'язково позначають стрілкою. Лінію потоку зазвичай підводять до середини блока. Відстань поміж паралельними лініями потоків має бути не меншою за 3 мм, поміж іншими блоками й символами – не менше за 5 мм. Лінію потоку можна обривати, використовуючи на місці обриву з'єднувачі, якщо схему виконано на двох чи більшій кількості аркушів, чи якщо з'єднувані блоки розташовано на значній відстані один від одного.

Рекомендується не перетинати лінії потоку, а використовувати з'єднувач блоків. Для використання з'єднувача блоки мають бути попередньо пронумеровані, а сам з'єднувач має містити цифру – номер блока, з яким відбувається поєднання або номер блока, з якого відбувається поєднання. Інакше: при з'єднуванні блоків можливо використовувати спільний символ, який записується у з'єднувачі блоків.

Основні види блоків та їхні призначення наведено у табл. 1.3.

Таблиця 1.3

Конфігурація блоків

Назва блока	Графічний вид блока	Призначення у програмі
Початок-кінець		Початок-кінець чи вхід-вихід для підпрограм
Блок введення-виведення		Введення даних та виведення результатів обчислювання
Блок обчислювань		Обчислювання за формулами
Умовний блок (розгалуження)		Вибір напрямку виконання алгоритму залежно від виконання певних умов
Блок модифікацій (заголовок циклу)		Задає початкове й кінцеве значення та крок для параметра циклу
Посилання на іншу сторінку		Задає зв'язки поміж частинами схеми, розташованими на різних сторінках
З'єднувач		Зазначення зв'язку поміж перерваними лініями потоку, що зв'язують блоки
Коментар		Зв'язок поміж елементом схеми і поясненням

Розмір a має бути кратним до 5, наприклад: $a = 20$ мм. Розмір $b = 1.5a$.

Перевагою блок-схем є те, що за їхньої допомоги можна наочно зобразити структуру алгоритму цілковито, окресливши його логічну сутність. Особливо це є важливим для задач економічного характеру і задач управління. Такі задачі містять чималу кількість операцій порівнювання, логічних, арифметичних та інших операцій, а отже, доволі складно встановити їхню послідовність у перебігу розв'язування задачі.

Запис алгоритму алгоритмічною мовою програмування має бути зрозумілим не лише людині, а й комп'ютерові, а тому потребує точного дотримання правил цієї мови. Такий спосіб подавання алгоритму буде розглянутий далі при вивченні мови програмування C++.

Програма – упорядкована послідовність команд, які належить виконувати. Принцип програмного керування, який є основним принципом будови всіх сучасних ЕОМ, за Дж. фон Нейманом, полягає у тому, що всі обчислення, відповідно до алгоритму розв'язання задачі, мають бути подані у вигляді програми. Кожна команда програми містить вказівки стосовно конкретної виконуваної операції, місця розташування (адреси) операндів та низки службових ознак. **Операнди** – змінні, значення яких беруть участь в операціях опрацювання даних. Список усіх змінних (початкових даних, проміжних значень та результатів обчислень) є ще одним неодмінним елементом будь-якої програми.

Для доступу до програм, команд та операндів використовуються їхні адреси. За адреси слугують номери комірок пам'яті ПК, призначених для зберігання об'єктів.

1.4 Види базових алгоритмів

Базові алгоритми організації обчислень поділяють на три основні види:

- ✓ лінійні (послідовності);
- ✓ розгалужені;
- ✓ циклічні.

Переважно вони є окремими частинами обчислювального процесу, тоді як загальний обчислювальний процес має складнішу (комбіновану) структуру. Зазвичай при написанні програми ці базові алгоритми поєднуються в такій структурі.

Лінійні алгоритми містять операції, які виконуються послідовно у порядку їхнього запису. Типовим прикладом такого алгоритму є стандартна обчислювальна схема, яка складається з трьох етапів:

- ✓ введення початкових даних;
- ✓ обчислення за формулами;
- ✓ виведення результату.

Розгалужені алгоритми для здобуття кінцевого результату містять команди вибору одного з кількох можливих напрямів обчислень (гілок) залежно від результату перевірки певної умови. Напрямок обчислень вибирається перевіркою, внаслідок якої можливі два випадки:

- ✓ «Так» – умову виконано;
- ✓ «Ні» – умову не виконано.

Циклічні алгоритми містять послідовність дій, які багаторазово повторюються. Цикли застосовують, коли для розв'язання задачі виникає потреба повторення низки кроків. Це призводить до багаторазового виконання одних і тих самих операторів у програмі за різних значень змінних. Керування повторенням циклу здійснюється за допомогою змінної, яка називається *параметром циклу*. Спочатку цьому параметру присвоюється певне початкове значення. Потім цикл виконується зі зміною параметра за кожного повторення від початкового до кінцевого значень на величину, що називається *кроком циклу*. Крок циклу може бути додатним чи від'ємним. Залежно від цього параметр циклу зростає чи зменшується. Цикл припиняється, якщо параметр циклу має значення, яке лежить поза межами діапазону між початковим і кінцевим значеннями.

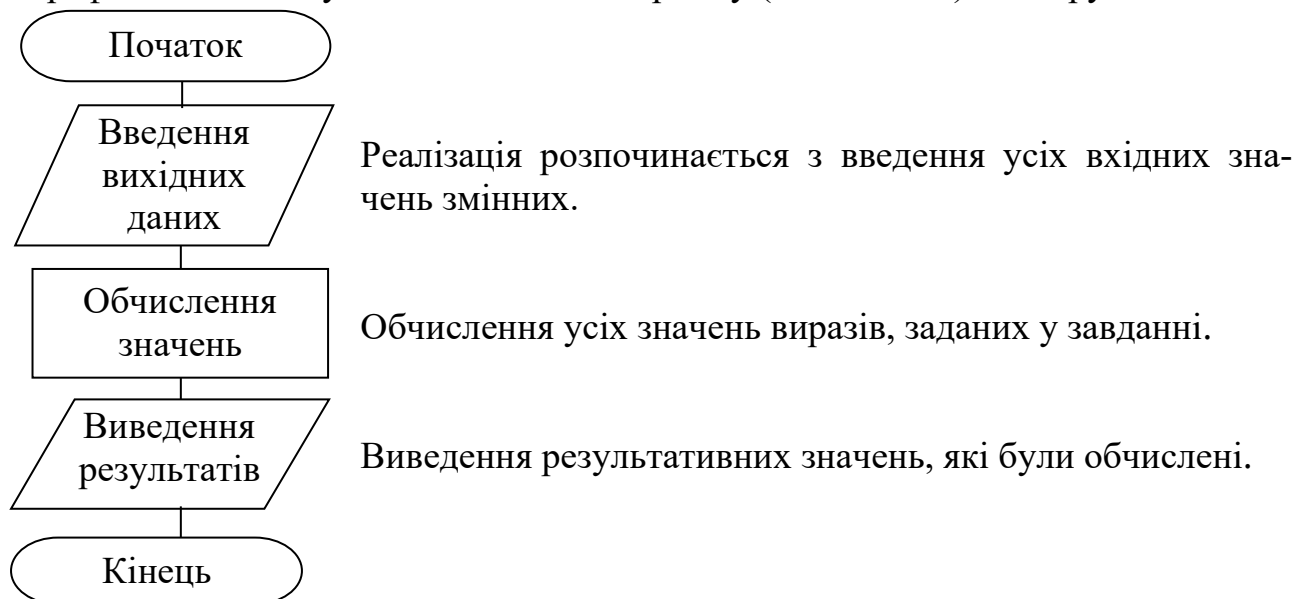
Розрізняють три базових алгоритмічних структури: лінійну (послідовність), розгалужену та циклічну (повторювання). Алгоритмічні структури будь-якої складності утворюються поєднанням послідовності цих базових алгоритмів. Приклади базових алгоритмічних структур наведено у підрозд. 1.5, а більш детально різноманітні схеми алгоритмів та їхні реалізації детально будуть розглянуті у наступних розділах.

1.5 Приклади базових алгоритмічних структур

1.5.1 Специфіка алгоритмів лінійної структури

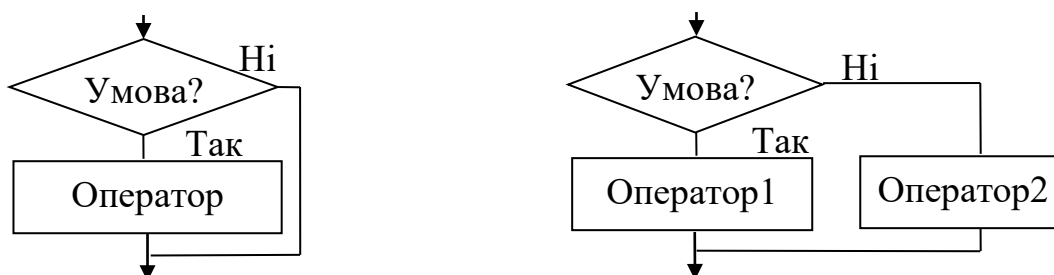
Лінійні алгоритми використовуються в структурах, де одна дія виконується слідом за іншою послідовно і при цьому жодна з дій не пропускається і не повторюється.

Словесний опис послідовності дій лінійної структури наведено праворуч, а графічне подання у вигляді схеми алгоритму (блок-схеми) – ліворуч.



1.5.2 Специфіка алгоритмів розгалуженої структури

Розгалужені алгоритми мають певну умову, залежно від істинності або хибності якої будуть виконуватись ті чи інші команди, при цьому деякі дії можуть не виконуватися взагалі (пропускатися).



Наведені два фрагменти схем алгоритмів відрізняються наявністю другого оператора, який виконуватиметься вразі невиконання (хибності) умови. У програмуванні ця гілка реалізується через ELSE.

1.5.3 Алгоритми циклічної структури

Циклічні алгоритми при кожному черговому виконанні циклу перевіряють певну умову на продовження роботи і, якщо умова істинна, цикл виконуватиметься, а, якщо умова хибна, відбудеться вихід з циклу. Тобто, **цикл** – це частина алгоритму, яка повторюється.

Перевірка умови може бути організована на початку циклу, і такий цикл називається циклом з передумовою (рис. 1.1, а), або наприкінці циклу – тоді такий цикл називається циклом з післяумовою (рис. 1.1, б). Різниця між такими циклами полягає в тому, що цикл з післяумовою виконується хоча б один раз, а цикл з передумовою може не виконуватися жодного разу.

Цикл з параметром (лічильником) (рис. 1.1, в) характерний тим, що заздалегідь відома або кількість повторювань циклу, або нижня та верхня межа значень і крок, з яким має змінюватись лічильник. Такий цикл виконуватиметься, доки параметр циклу лежатиме у межах від початкового до кінцевого значення.

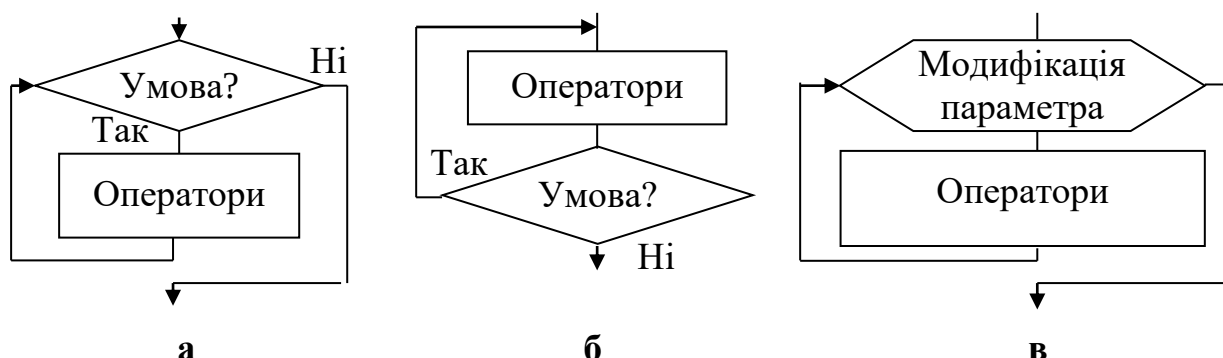


Рис. 1.1. Різновиди схем алгоритмів циклічної структури:

- а – цикл з передумовою;
- б – циклом з післяумовою;
- в – цикл з параметром (лічильником)

Питання та завдання для самоконтролю

- 1) З яких етапів складається процес розроблення програми?
- 2) Що таке алгоритм?
- 3) Назвати основні властивості алгоритмів.
- 4) Які є способи подання алгоритмів?
- 5) Який спосіб задавання алгоритмів є найпоширенішим?
- 6) Як називається спосіб подання алгоритмів, де кожний його крок подано спеціальним блоком, який показує виконувану дію?
- 7) Що таке схема алгоритму (блок-схема)?
- 8) Яка перевага блок-схем?
- 9) Перелічити блоки, використовуванні в схемах алгоритму.
- 10) Які правила поєднання блоків поміж собою лініями потоку?
- 11) Який напрямок ліній потоку не позначають стрілками?
- 12) Коли треба лінії потоку треба позначати стрілками?
- 13) До якої частини блока слід підводити лінії потоку при поєднанні блоків?
- 14) Які відстані мають бути поміж паралельними лініями потоків?
- 15) Які відстані мають бути поміж блоками ?
- 16) Яких пропорції між шириною та висотою блоків слід дотримуватись?
- 17) Як виглядає блок для початку і кінця блок-схеми? Чому є помилковим використання овалу для цього блока?
- 18) Як виглядає умовний блок для організації розгалуження?
- 19) Яка форма блоків для введення-виведення даних?
- 20) Як виглядає блок модифікації для організації циклу з параметром?
- 21) Яка роль блока з'єднувача?
- 22) Які види з'єднувачів існують? Чим вони відрізняються?
- 23) Які значення ставляться у блоках з'єднувачах?
- 24) Назвати базові алгоритми організації обчислень.
- 25) Охарактеризувати алгоритми лінійної структури.
- 26) У чому полягає специфіка алгоритмів з розгалуженнями?
- 27) Що таке цикл?
- 28) Охарактеризувати різновиди схем алгоритмів циклічної структури.
- 29) Коли використовується блок модифікації у схемах алгоритму?
- 30) Що таке програма?

РОЗДІЛ 2

ЗНАЙОМСТВО З C++.

СПОСОБИ ВВЕДЕННЯ-ВИВЕДЕННЯ ДАНИХ

2.1 Технологія створювання інтерфейсних програм у Visual C++

2.1.1 Засоби системи програмування Visual C++

Кожна програма повинна мати зручний інтерфейс для спілкування з користувачем. Основним елементом інтерфейсу у Windows є вікна. Одним з різновидів вікон є форма, яка може містити кнопки, текстові поля, перемикачі тощо. Тому програми, написані для використання у Windows, зазвичай мають інтерфейс, подібний до вікон та форм. Для швидкого і зручного створювання програм з графічним інтерфейсом використовують спеціальні середовища візуального програмування. Майже кожна сучасна мова програмування має принаймні одне таке середовище: Basic – Visual Basic, C++ – Microsoft Visual C тощо. Візуальне програмування ще називають Rapid Application Development (RAD), «швидке розроблення програмних проектів». Технологія RAD суттєво прискорює створення програм з графічним інтерфейсом.

Інтегроване середовище розроблення Microsoft Visual Studio є це набором інструментів для створення програмного забезпечення. Ці інструменти призначені для максимально ефективної спільної роботи; всі вони доступні в інтегрованому середовищі розроблення (IDE – Integrated Development Environment) Visual Studio.

Visual Studio можна використовувати для створення різних типів застосунків, від простих програм та ігор для мобільних клієнтів до великих і складних систем, які обслуговують підприємства та центри оброблення даних. При цьому можна створювати програмні застосунки та ігри не лише для платформи Windows, а й для Android та iOS, веб-сайти і веб-служби на основі ASP.NET, JQuery, AngularJS та інших популярних платформ¹ тощо.

Залежно від версії Visual Studio забезпечує підтримку C#, C++, JavaScript, F# і Visual Basic. Visual Studio добре працює й інтегрується зі сторонніми застосунками, наприклад, Unity і Apache Cordova, за допомогою відповідних розширень.

Система Visual Studio, подібно до інших систем візуального програмування, насамперед є посередником поміж інтерфейсом прикладного програмування Windows (API – Application Program Interface) та програмістом, надаючи змогу навіть програмістам-початківцям оперативно створювати програмні проекти, які матимуть графічний інтерфейс користувача (GUI – Graphic User Interface)

¹ Інтегроване середовище розроблення Visual Studio. URL: <https://msdn.microsoft.com/ru-ru/library/dn762121.aspx> (дата звернення 7.10.2018).

найрізноманітнішої спрямованості, від суто обчислювальних і логічних до графічних і мультимедійних.

Мова програмування Visual C++ і засоби розроблення дозволяють розробляти універсальні програмні проекти Windows некерованого коду, класичні та серверні програми некерованого коду, міжплатформові бібліотеки для Android, iOS і Windows, а також керовані програми, що працюють на платформі .NET Framework.

.NET Framework – це платформа розроблення для створення програмних застосунків для інтернету, Windows, Windows Phone, Windows Server і Microsoft Azure. Вона складається із середовища CLR і бібліотеки класів .NET Framework, яка містить великий набір функціональних можливостей, а також забезпечує підтримку багатьох галузевих стандартів. Платформа .NET Framework надає багато сервісів, включаючи керування пам'яттю, функції безпеки роботи з мережами тощо. Вона надає зручні структури даних та інтерфейси API, які абстрагують операційну систему Windows нижчого рівня. У .NET Framework можна використовувати різні мови програмування, включаючи C#, F# і Visual Basic².

У кожному версію платформи .NET Framework входить середовище виконання коду CLR, бібліотеки базових класів та інші керовані бібліотеки. Код, розроблений з мовним компілятором, який звертається до загальномовного середовища виконання CLR, називають керованим кодом. У ньому використовуються переваги таких засобів, як об'єднання мов програмування, спільне опрацювання винятків, посилена безпека, спрощена модель взаємодії компонентів тощо.³

Отже, середовище CLR спрощує розроблення компонентів і програм, об'єкти яких можуть працювати в різних мовах. Об'єкти, написані різними мовами, можуть взаємодіяти один з одним, а їхня поведінка може бути тісно інтегрованою. Наприклад, розробник може визначити клас, а потім на іншій мові створити похідний від нього клас або викликати метод з вихідного класу. Можна також передати екземпляр класу у метод класу, написаний іншою мовою. Така інтеграція мов програмування можлива в силу того, що мовні компілятори і програми, які звертаються до середовища виконання, використовують систему загальних типів, визначену середовищем виконання, і дотримуються правил середовища виконання при визначенні нових типів, а також при створенні, використанні, збереженні і прив'язках до типів.

Visual C++ є частиною комплексу Microsoft Visual Studio, яка дозволяє розробляти як консольні програми, так і програми з графічним інтерфейсом. Мова Visual C++ є унікальним серед інших мовних засобів, що надаються середовищем Visual Studio, позаяк вона підтримує розроблення програм як для виконання у середовищі «чистої» Windows, так і на керованому (managed) C++.

² Керівництво по .NET Framework. URL: <https://docs.microsoft.com/ru-ru/dotnet/framework/index> (дата звернення 7.10.2018).

³ Огляд середовища CLR. URL: <https://docs.microsoft.com/ru-ru/dotnet/standard/clr> (дата звернення 7.10.2018).

Керований⁴ програмний код потребує та виконується винятково віртуальною машиною Common Language Runtime (скорочено CLR – «загальне середовище виконання мов», яке є компонентом пакета Microsoft .NET Framework), на якій виконуються всі мови платформи .NET Framework.

Далі буде розглянуто такі засоби створення програмних проектів:

- консольної програми Win32;
- консольної програми CLR;
- програми Windows Forms.

2.1.2 Послідовність створення консольної програми Win32 у Visual C++

Консольний застосунок – це програма, для якої пристроєм введення інформації є клавіатура, а пристроєм виведення – монітор у символьному режимі.

Консольний режим – це режим створювання програм користувача без використання графічного інтерфейсу, а з відображенням лише символьної інформації. В операційній системі робота у консольному режимі здійснюється засобами командного рядка.

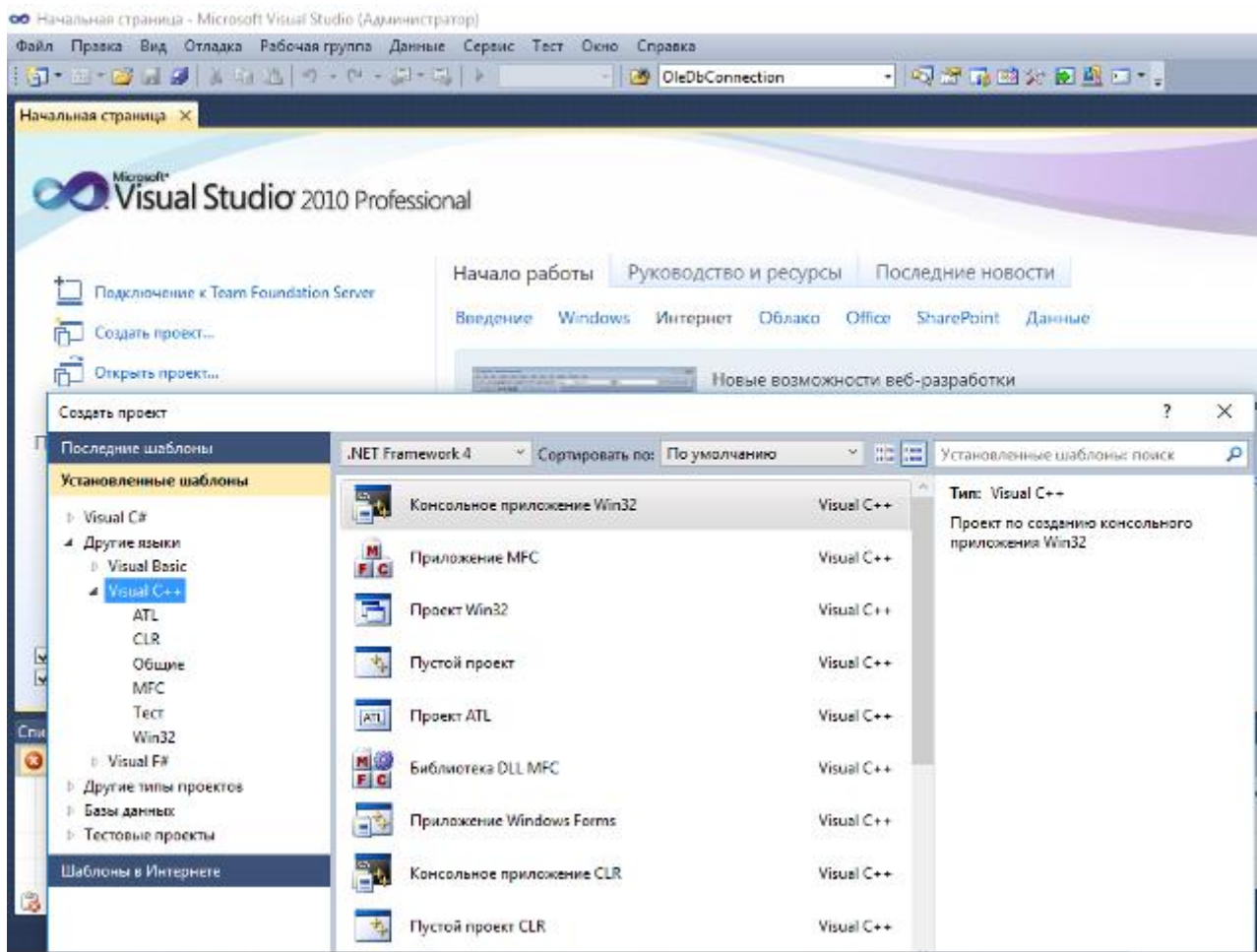
Виконання консольної програми відбувається у чорному консольному екрані. Для введення даних слід використовувати спеціальні команди. Під час виконання вони скомандують програмі зупинитися й очікувати, допоки користувач введе значення і натисне клавішу [Enter]. Виведення даних виконується у теж саме чорне вікно і завжди потребує вказівок та пояснень для користувача.

Виведення означає, що певний текст чи то значення певних змінних висвітяться на екрані і користувач зможе їх прочитати. Введення означає, що користувач задає значення змінної, яке буде використано у програмі.

Для створення консольної програми слід після запуску Visual Studio, коли відкриється початкова сторінка, виконати команду *Создать проект*, що призведе до відкриття діалогового вікна *Создать проект*. Після вибору мови C++ слід вибрати вид створюваного програмного проекту *Win32 / Консольное приложение Win32*.

Не закриваючи вікно *Создать проект*, слід ввести ім'я програмного проекту, при цьому для вибору папки для зберігання можна скористатися кнопкою *Обзор*. Після цього натиснути *ОК*, що призведе до появи діалогового вікна майстра програм Win32, в якому слід просто натиснути кнопку *Готово*.

⁴ Слово керований тут стосується методу обміну інформацією між програмою і виконавчим середовищем. Воно означає, що в будь-якій точці виконання, виконавче середовище може припинити виконання й отримати інформацію, специфічну для поточного стану.



Після цього відкриється вікно сpp-файлу програмного проекту з автоматично згенерованим порожнім шаблоном головної функції:

```
int main()
{ return 0;
}
```

Тепер можна приступити до написання програмного коду.

Як приклад наведемо програмний код обчислення суми двох чисел:

```
#include <iostream>
using namespace std;
int main()
{ setlocale(0, ".1251");
  double a,b;
  cout << "Введіть два числа (через пробіл)\n";
  cin >> a >> b;
  cout << "Сума цих чисел - " << a + b << endl;
  system ("pause>>void");
  return 0;
}
```

Узагальнене тлумачення програмного коду розглянемо дещо нижче у підрозд. 2.2, а зараз наведемо порядок запуску програм на виконання.

Запустити програму можна одним зі способів:

- командою меню *Отладка / Запуск без отладки*;
- натисканням клавіш [Ctrl] + [F5].

Результатом роботи програми буде поява вікна консолі з таким текстом (при натисканні будь-якої клавіші вікно закриється):

```
Введіть два числа (через пробіл)  
129.5 0.75  
Сума цих чисел - 130.25
```

Ще одним способом запуску програми є запуск налагоджувача:

- командою меню *Отладка / Начать отладку*;
- кнопкою  на панелі інструментів;
- натисканням клавіші [F5].

Відмінність цього способу буде видно лише за відсутності у програмному коді останньої команди, яка організує паузу з очікуванням натискання будь-якої клавіші. До речі, останню команду `system("pause>>void")` можна змінити парою команд:

```
cin.get(); cin.get();
```

Ще однією суттєвою відмінністю запуску програми у налагоджувачі є можливість встановлення точок зупину, скориставшись клавішею [F9] чи то командою *Отладка / Точка останова*. При встановленні такої точки, після запуску програми на виконання, вона зупиниться на відповідній команді, і можна буде відслідкувати значення тієї чи іншої змінної на певному етапі виконання програми. Найпростішим способом здійснення цього є наведення покажчика миші на змінну, при цьому біля курсора сформується віконце з поточним значенням цієї змінної.

Тепер, як було обіцяно, наведемо узагальнене тлумачення вищенаведеного програмного коду, а більш глибоке розуміння Ви набудете при подальшому вивченні відповідних програмних засобів.

На початку програми для можливості коректного використання операторів введення-виведення даних `cin` та `cout` у програмному коді було долучено бібліотеку `iostream`. Крім того, оскільки ці команди належать простору імен `std`, було прописано використання цього простору – `using namespace std`.

В основній програмі `_tmain()` на початку для можливості коректного відображення символів кирилиці було долучено використання кодування 1251. Далі оголошено та введено значення двох дійсних (тип `double`) змінних `a` та `b`. Після цього виводиться відповідний коментар та викликається функція `nod()`, числовий результат якої виводиться. Наприкінці організовується пауза (`system("pause>>void")`) як очікування натискання будь-якої клавіші.

Зберегти всі файли програмного проекту можна командою *Файл / Сохранить все*, чи то натисканням клавіш [Ctrl]+[Shift]+[S], чи то кнопкою  на панелі інструментів.

До речі, змінити фон і колір виведеного тексту в консолі можна використовуючи функцію `system`, в яку передати рядок: `"color <A><B>"`, де `<A>` і `<B>` – шістнадцяткові цифри – перша задає колір фону, а друга – колір переднього плану (колір шрифту).

Значення цифр:

0 – чорний	5 – ліловий	A – світло-зелений
1 – синій	6 – жовтий	B – світло-блакитний
2 – зелений	7 – білий	C – світло-червоний
3 – блакитний	8 – сірий	E – світло-жовтий
4 – червоний	9 – світло-синій	F – яскраво-білий

Приклад:

```
system("color F0"); // Встановити білий фон і чорний текст
```

2.1.3 Створення консольної програми CLR у Visual C++

CLR – це розширення C++, введене фірмою Microsoft, яке керує виконанням коду, пам'яттю, потоками і роботою з віддаленими комп'ютерами, при цьому забезпечуючи безпеку і надійність виконання коду.

Відмінність послідовності створення цього шаблону програмного проекту від попереднього полягає у використанні команди *Создать проект / CLR / Консольное приложение CLR*. Після зазначення імені проекту та натискання кнопки *ОК* з'явиться вікно сpp-файлу програмного проекту з таким автоматично згенерованим шаблоном головної функції:

```
using namespace System;

int main(array<System::String ^> ^args)
{
    Console::WriteLine(L"Здравствуй, мир!");
    return 0;
}
```

У цій функції вже існує долучення простору імен System та команда виведення «вітального» рядка WriteLine(). Тобто для введення-виведення даних у CLR-проектах можна використовувати як команди C++, так і методи простору імен System, а саме: ReadLine(), WriteLine() для читання та виведення рядка символів та ін. Отже, у такому програмному проекті можна реалізувати як програмний код, наведений на с. 18, так і модифікувати його до такого вигляду:

```
using namespace System;

int main(array<System::String ^> ^args)
{
    double a,b;
    Console::Write("Введіть два числа (через Enter)\n");
    a = Convert::ToDouble(Console::ReadLine());
    b = Convert::ToDouble(Console::ReadLine());
    Console::WriteLine("Сума цих чисел - {0:f}", a+b);
    Console::Read();
    return 0;
}
```



```
Введіть два числа (через Enter)
34,67
-2,5
Сума цих чисел - 32,17
```

2.1.4 Створення програмного проекту Windows Forms у Visual C++

Для створення програмного проекту з формою слід запустити Visual Studio та виконати команду *Создать проект / Visual C++ / CLR / Приложение Windows Forms*. Не закриваючи вікно *Создать проект*, натиснути кнопку *Обзор*, вибрати папку для зберігання та задати ім'я програмного проекту, після чого натиснути кнопку *ОК*. Після зазначених дій відбудеться створення програмного проекту та відкриється вікно конструювання його форми (рис. 2.1).

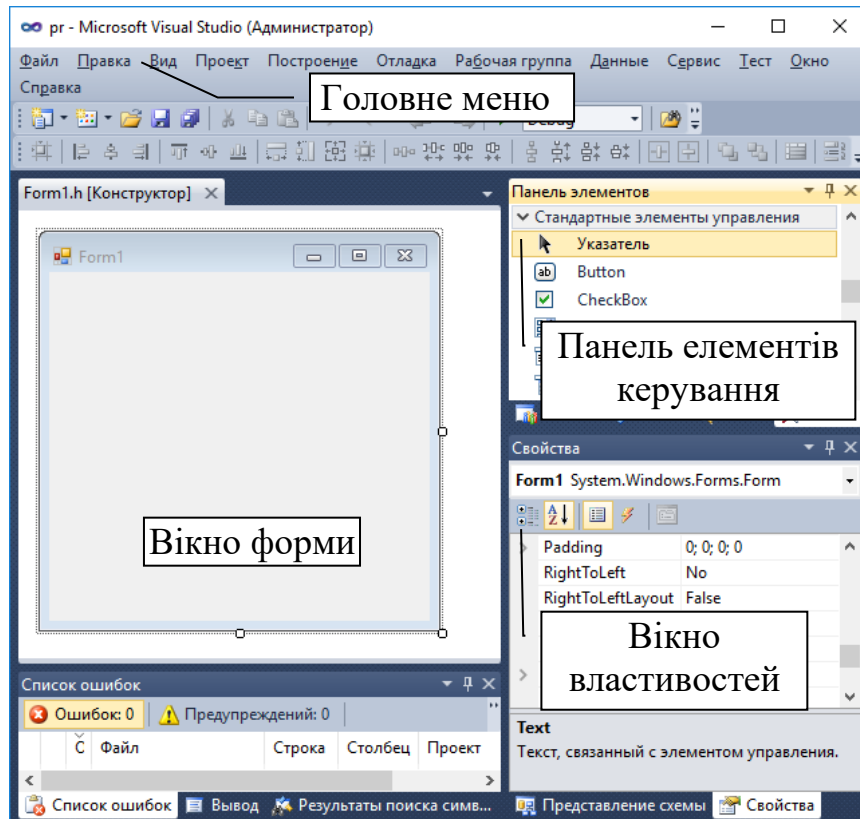


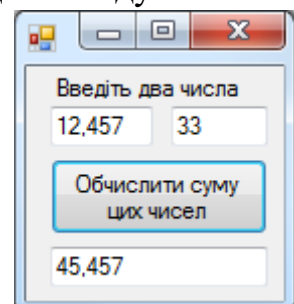
Рис. 2.1. Вигляд вікна конструювання форми проекту Visual C++

Для налаштування форми слід на ній розмістити необхідні елементи, наприклад: три елементи `textBox`, один `button` та один `label`. Також треба задати значення властивості `Text` елементів `label1` та `button1`, увівши в неї відповідний текст. Після подвійного клацання по командній кнопці `button1` система сформує порожній шаблон функції `button1_Click`:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
}
```

Після введення програмного коду шаблон функції набуде вигляду:

```
private: System::Void button1_Click(System::Object^ sender,
                                     System::EventArgs^ e)
{
    double a, b;
    a = Convert::ToDouble(textBox1->Text);
    b = Convert::ToDouble(textBox2->Text);
    textBox3->Text = Convert::ToString(a+b);
}
```



Звернімо увагу на те, що фігурні та круглі дужки мають у C++ різне призначення. По-перше, у круглих дужках записуються параметри функцій, а, по-друге, круглі дужки використовуються за потреби при записуванні арифметичних виразів. Фігурні дужки називають *операторними*; у C++ вони є аналогом `begin/end` інших мов. Відкритою фігурною (`{`) дужкою розпочинається кожна функція, у тому числі й функції відгуку на події, а закрита фігурна дужка (`}`) ставиться після останньої команди функції. Окрім того, фігурні дужки використовуються всередині функцій для поєднування групи команд щодо заданої команди, наприклад, якщо в циклі слід виконувати понад одну команду.

2.1.5 Специфіка конструювання проектів з формою

Вікно форми займає найбільше місце і є прямокутним сірим «контейнером» (див. рис. 2.1), на який при проектуванні форми розміщують елементи керування (кнопки, надписи, панелі, текстові вікна тощо). Форма сама є елементом з назвою `Form`. Без додаткових вказівок заголовок елемента керування (властивість `Text`) збігається з його назвою, до якої додається порядковий номер, розпочинаючи з 1 (наприклад, `button1`, `button2`). Але заголовок можна змінити за допомогою властивості `Text`. Розміри форми також можна змінювати чи то через панель *Properties* (*Свойства*), чи просто «зачепивши» мишкою за лінію межі форми (цього моменту курсор набуває вигляду двонапрямленої стрілки).

Після конструювання візуального інтерфейсу засобами форми розробник приступає до написання програмного коду, виконання команд якого забезпечить розв'язання певної задачі. Для переходу до *вікна редактора коду* (рис. 2.2) програми для написання тексту програми слід натиснути клавішу `[F7]`. Зазвичай вікно редактора коду перекривається вікном форми. Перемикання поміж вікнами форми й редактора можна здійснювати чи то натисканням миші на видимій частині вибраного вікна, чи то клавішами `[F7]` та `[Shift]+[F7]`. При цьому найпоширенішим засобом створення шаблону функції, наприклад, `button1_Click`, й одночасного переходу до вікна редактора коду є подвійне клацання по командній кнопці `button1`.

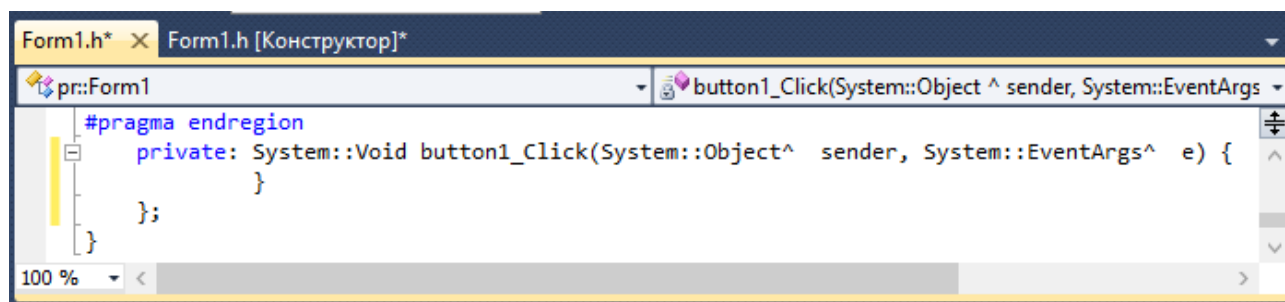


Рис. 2.2. Вигляд вікна редактора коду

Вікно редактора коду для проекту з формою навіть для порожньої форми при першому відкриванні вже міститиме певний програмний код. При створенні проекту середовище програмування Microsoft Visual Studio автоматично генерує вихідний код, що містить власний простір імен (`namespace` з ім'ям проекту) та головний клас `Form1`:

```
public ref class Form1 : public System::Windows::Forms::Form
```


Фігурні дужки саме цього класу та простору імен з ім'ям проекту є останніми закривними операторними дужками у проекті і видалення їх призведе до помилок компіляції.

Крім того, початковий програмний код містить визначення бібліотечних просторів імен, які планується використовувати у програмі:

```
using namespace System;  
using namespace System::ComponentModel;  
using namespace System::Collections;  
using namespace System::Windows::Forms;  
using namespace System::Data;  
using namespace System::Drawing;
```

Узагальнено така програма являє собою сукупність класів, що інкапсулюють необхідні для розв'язання деякої задачі властивості, методи та події.

Класи програми, у тому числі і головний клас, слід розміщувати у власному просторі імен з іменем проекту, який оголошується за допомогою ключового слова `namespace`. Використання власних просторів імен гарантує однозначний та зрозумілий розподіл ідентифікаторів класів. Взагалі, програма може містити довільну кількість просторів імен, в тому числі і вкладених.

У вікно редактора коду, як і в інші вікна Visual C++, вбудовано контекстну довідку. Щоб отримати довідку стосовно певного слова (терміну) коду, достатньо поставити курсор на це слово і натиснути клавішу [F1].

Головне меню міститься у верхній частині основного вікна (див. рис. 2.1). Воно надає можливість доступу до всіх інструментальних засобів створювання, модифікації та керування проектом. Меню містить багато підменю: *File (Файл)*, *Edit (Правка)*, *Save All (Зберегти все)* тощо.

Підменю **Edit (Правка)** містить команди редагування, роботи з областю обміну даними, відміни дій і керування відображенням елементів керування. Його основними опціями є:

- Undo, Redo* – відмінити чи відновити останню зміну (еквівалентні дії викликаються комбінаціями клавіш: [Ctrl]+[Z] і [Shift]+[Ctrl]+[Z]);
- Cut, Copy* – вирізати чи лише скопіювати елементи керування чи фрагмент тексту до буфера ([Ctrl]+[X] і [Ctrl]+[C]);
- Paste* – вставити з буфера дані ([Ctrl]+[V]), які були до цього скопійовані чи вирізані в буфер;
- Delete* – вилучити виокремлені елементи керування на формі чи текст (клавіша [Delete]);
- Select All* – виокремити всі елементи керування на формі чи увесь текст поточного модуля ([Ctrl]+[A]);
- Search* – команди пошуку ([Ctrl]+[F]) і заміни ([Ctrl]+[H]) фрагментів тексту, а також покрокового пошуку рядків з помилками компілювання.

Для решти підменю головного меню зазначимо лише область їхньої дії, позаяк при початковому вивченні Visual C++ та створенні нескладних проектів

можна користуватися значеннями опцій за промовчанням чи використовувати «гарячі» кнопки чи то комбінації клавіш клавіатури.

Підменю **View** (*Вид*) містить команди для відображення панелі елементів керування, вікна властивостей, оглядача розв'язків, панелі з помилками тощо.

Підменю **Project** (*Проект*) містить команди конструювання проектів, а також встановлення опцій поточного проекту.

Підменю **Run** (*Отладка*) містить команди для запуску проекту на виконання ([F5]), засобів контролю покрокового виконання і налагоджування програм.

Підменю **Help** (*Справка*) містить команди для виклику різних розділів довідкової системи.

«Гарячі» кнопки інструментальних панелей призначені для швидкого запуску найчастіше використовуваних команд головного меню (див. рис. 2.3).



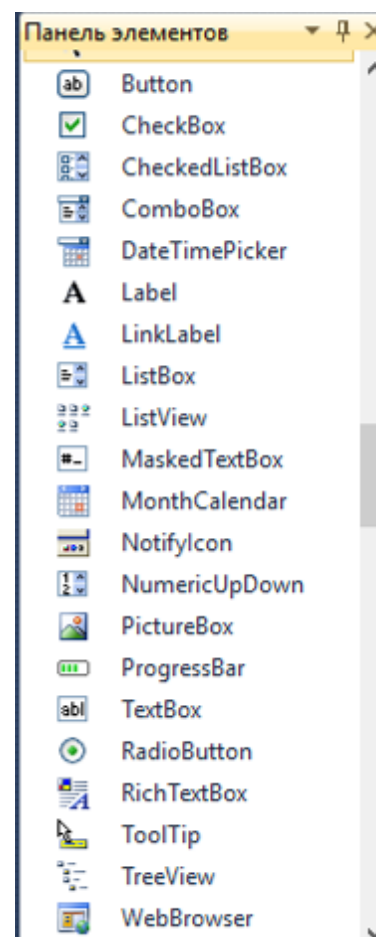
Рис. 2.3. Піктографічні кнопки

Клацання лівою кнопкою миші на якій завгодно «гарячій» кнопці спричинить виконання відповідної функції головного меню. Функції основних «гарячих» кнопок наведені у табл. 2.1. Кнопки піктографічного меню можна редагувати, тобто вилучати й долучати нові кнопки.

Таблиця 2.1

Функції «гарячих» кнопок

Кнопка	Назва та призначення
	Створити новий проект [Ctrl]+[Shift]+[N]
	Додати новий елемент [Ctrl]+[Shift]+[A]
	Відкрити файл [Ctrl]+[O]
	Зберегти файл [Ctrl]+[S]
	Зберегти усі файли проекту [Ctrl]+[Shift]+[S]
	Вирізати [Ctrl]+[X]
	Копіювати [Ctrl]+[C]
	Вставити [Ctrl]+[V]
	Відмінити [Ctrl]+[Z]
	Повернути [Ctrl]+[Y]
	Назад [Ctrl]+[-]
	Вперед [Ctrl]+[Shift]+[-]
	Компілювати і виконати програму [F5]



Панель елементів керування сформовано у вигляді сукупності елементів. Ця панель відкривається клавішами [Ctrl] + [W], [X]. Щоб отримати доступ до певного елемента керування, слід вибрати потрібний елемент і розмістити його на формі. Елементи керування на панелі подано у вигляді кнопок. У таблиці 2.2 розглянемо найпоширеніші у використанні елементи керування.

Таблиця 2.2

Найпоширеніші у використанні елементи керування

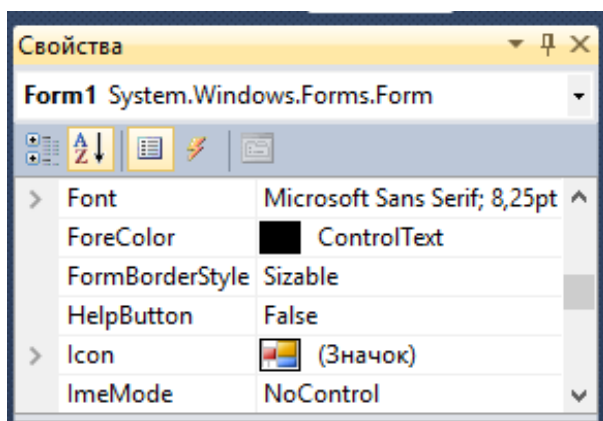
Кнопка	Назва	Призначення
	button (кнопка)	Прямокутна кнопка, натискання якої ініціює задані дії у програмі; надпис на кнопці задається властивістю Text
	label (надпис – позначка)	Надпис на формі, текст якого задається властивістю Text. Зазвичай використовується для пояснювальних надписів. Властивість AutoSize є ознакою того, що розмір поля визначається його вмістом. Властивість TextAlign визначає спосіб вирівнювання тексту
	textBox (текстовий редактор)	Відображення, введення та виведення тексту; цей текст зберігається у властивості Text. Параметри шрифту: назва, розмір, колір, стиль визначається за допомогою властивості Font. Значення True властивості Multiline надає можливість перенесення тексту по словах. Значення True властивості WordWrap надає можливість перенесення тексту по словах
	richTextBox (редактор тексту з розширеними можливостями)	Багаторядкове текстове вікно для відображення, введення чи виведення тексту та значень даних програми; текст зберігається у властивості Text
	listBox (список вибору)	Список текстових варіантів для вибору. Елементи списку містяться у властивості Items, а номер вибраного під час виконання програми елемента – у властивості SelectedIndex. Можна динамічно долучати, вилучати, вставляти і переміщувати елементи списку за допомогою методів Add, Clear, Insert та Remove об'єкта Items. Значення True властивості Sorted встановлює сортування елементів списку за абеткою
	comboBox (комбінований список)	Список елементів (властивість Items) розкривається (випадає) для вибору. Значення вибраного елемента надається властивості Text. Елемент поєднує в собі два елементи керування: ListBox та textBox. Значення True властивості Sorted встановлює сортування елементів списку за абеткою
	tabControl (колекція вкладок)	На різних вкладках цього елемента керування можна розташовувати інші елементи керування. Кількість вкладок, надписи на них та інші їхні властивості задаються властивістю TabPages

Закінчення табл. 2.2

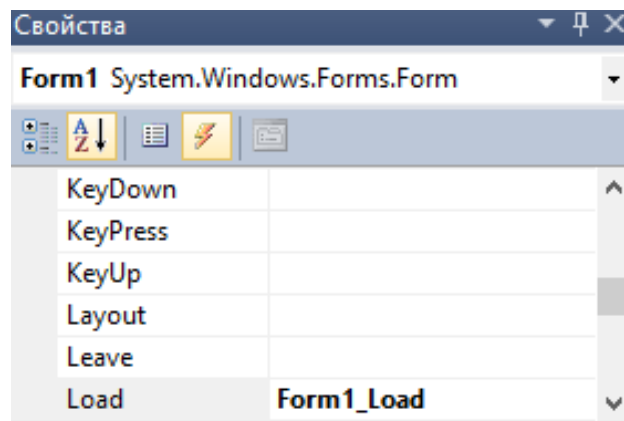
Кнопка	Назва	Призначення
	dataGridView (таблиця рядків)	Таблиця для відображення текстової інформації в її рядках та стовпцях. Властивість Cells дозволяє програмне звертання до комірок елементом керування. Колекція властивості Columns задає властивості для кожного зі створюваних стовпців. Рядки можна долучати лише у програмі
	menuStrip (меню)	Панель команд меню. Ідентифікатори всіх команд меню визначаються властивістю Items, яка має доступ до кожної команди меню
	checkBox (індикатор з прапорцем)	Квадратний індикатор дозволяє користувачеві вмикати/вимикати опції програми, оскільки елемент керування допускає два стани: «ввімкнено» і «вимкнено» (властивість Checked)
	radioButton (радіокнопка)	Кругла кнопка з двома станами і текстом опису, який специфікує її призначення. Радіокнопки – набір взаємовиключних варіантів вибору: лише одна кнопка може бути вибрана на даний момент часу (позначається внутрішнім чорним кружечком), а з попередньо вибраної кнопки вибір автоматично знімається. При натисканні радіокнопки властивість Checked змінюється і виникає подія OnClick

Якщо вибрати будь-який елемент керування та розмістити на формі, його властивості можна побачити та змінити у вікні властивостей.

Вікно властивостей (*Properties*) забезпечує простий і зручний інтерфейс для змінювання значень властивостей елементів керування і. Це вікно надає доступ не лише до властивостей, а й до керування подіями, на які реагують елементи, якщо клацнути кнопку на ньому  **Events** (події) (рис. 2.4).



а



б

Рис. 2.4. Вікна властивостей:

а – сторінка властивостей (*Properties*); б – сторінка подій (*Events*)

Значення властивостей можна змінювати. Приміром, якщо клацнути на формі, а тоді вибрати її властивість Text, то у вікні зі значенням цієї властивос-

ті `Form1` можна ввести новий текст, який і стане заголовком форми. Властивість `BackColor` має список можливих значень, який можна відкрити, клацнувши кнопку  у вікні цієї властивості.

Деякі з властивостей мають складну структуру. Клацання миші у полі значень властивості призводить до появи у цьому полі кнопки . Після натискання цієї кнопки відкриється діалогове вікно, котре і дозволяє встановлювати нові значення цієї властивості. Наприклад, властивість `Font` дозволяє задавати атрибути тексту: шрифт, розмір, стиль та колір літер (рис. 2.4, б).

Усі властивості мають за промовчанням певні значення при створенні елемента керування. Наприклад, для елемента керування `label` властивість `AutoSize` (автомасштаб) має значення `True` (істина) без додаткових вказівок. Це означає, що розмір елемента підлаштовується під розмір його вмісту. Але можна задавати потрібний розмір за допомогою властивості `Size` (довжина та висота), якщо при цьому змінити значення властивості `AutoSize` на `False` (хибність).

Якщо на формі треба розмістити кілька однотипних елементів керування, то усталено їхніми іменами буде назва відповідного елемента, наприкінці якого зазначається його порядковий номер на формі. Наприклад, `button1`, `button2`, ... чи `textBox1`, `textBox2`, ...

Вкладка подій (*Events*) вікна властивостей показує список подій, доступних для того чи іншого елемента керування, і дозволяє визначати реакцію виокремленого елемента на ту чи іншу подію (програмування для операційних систем з графічним користувацьким інтерфейсом). Кожен елемент керування має свій власний набір можливих подій.

Найбільш поширеними у застосуванні подіями є такі:

- ✓ `Click` – клацання лівою кнопкою миші на об'єкті;
- ✓ `DoubleClick` – подвійне клацання лівою кнопкою миші на об'єкті;
- ✓ `TextChanged` – внесення змін до вмісту об'єкта;
- ✓ `Load` – подія відбувається при кожному завантаженні форми;
- ✓ `Activated` – подія відбувається при активації форми;
- ✓ `MouseDown` – натискання кнопки миші над елементом керування;
- ✓ `MouseUp` – відпускання кнопки миші над елементом керування;
- ✓ `MouseMove` – переміщення миші над елементом керування;
- ✓ `KeyPress` – натискання символної клавіші клавіатури;
- ✓ `KeyDown` – натискання будь-якої клавіші клавіатури;
- ✓ `KeyUp` – відпускання будь-якої клавіші клавіатури;
- ✓ `Enter` – подія відбувається в момент набуття елементом фокуса;
- ✓ `Leave` – подія відбувається в момент втрати елементом фокуса;
- ✓ `DragDrop` – подія відбувається в момент відпускання перетягнутого елемента керування над даним елементом.

Назви подій наведені у лівій колонці вкладки (рис. 2.4, б). Для того, щоби створити функцію відгуку на певну подію, треба вибрати на формі за допомогою миші необхідний елемент керування, відкрити вкладку подій вікна властивостей і двічі клацнути лівою клавішею миші на колонку поруч з цією подією. При цьому автоматично буде згенеровано шаблон функції відгуку на подію, відкриється редактор коду, в якому з'явиться текст порожньої функції, а курсор при цьому позиціонуватиметься всередині її операторних дужок `{ ... }` для подальшого введення програмного коду. Наприклад, для події `Click` елемента

керування `button1`, розміщеного на формі `Form1`, шаблон відповідної функції в `Form1.h` матиме вигляд:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
}
```

Багато властивостей і методів є притаманними більшості елементів, наприклад: ім'я, розміри, доступність, видимість тощо. У табл. 2.3 наведено найчастіше уживані властивості, а в табл. 2.4 – методи функціонування елементів.

Що стосується методів, вони є функціями, пов'язаними з елементами, і по суті є частиною цих об'єктів. Виклик методів і звертання до властивостей елементів здійснюють за допомогою операції вибору елемента «->», наприклад:

```
button1->Show();
```

Таблиця 2.3

Основні властивості стандартних елементів

Властивість	Пояснення	Елементи з такою властивістю
<code>AutoSize</code>	Автомасштаб: якщо встановити в 1 (True), розмір елемента автоматично адаптується до розміру символів тексту	<code>label</code> , <code>button</code> , <code>panel</code> та інші
<code>Text</code>	Текст	<code>button</code> , <code>label</code> та інші
<code>BackColor</code>	Колір елемента керування	Більшість видимих
<code>Enabled</code>	Доступність до елемента. Визначає, чи реагує елемент на події, пов'язані з мишею, клавіатурою і таймером. Значення <code>False</code> блокує доступ, а сам елемент набуває переважно більш світлого кольору	Усі видимі
<code>Font</code>	Параметри шрифту	<code>button</code> , <code>label</code> , <code>textBox</code> та інші
<code>Size</code>	Розміри (ширина та висота) елемента в пікселях. Задається через крапку з комою, наприклад: 40; 20. Тут перший розмір задає ширину 40 пікселів, а другий – ширину 20	Усі видимі
<code>SelectedIndex</code>	Номер вибраного елемента списку. Нумерація розпочинається з 0	<code>listBox</code>
<code>Items</code>	Список можливих значень списку	<code>listBox</code> , <code>comboBox</code> , <code>menuStrip</code> та інші
<code>Location</code>	Розташування елемента щодо верхнього лівого краю форми у пікселях. Верхній лівий кут форми має координати (0, 0). Задається через крапку з комою, наприклад: 40; 20. Тут перший розмір задає відступ ліворуч 40 пікселів, а другий – відступ згори 20	Усі видимі

Закінчення табл. 2.2

Властивість	Пояснення	Елементи з такою властивістю
Sorted	Якщо встановити в 1 (True), усі елементи списку буде відсортовано за зростанням	listBox, comboBox
Visible	Видимість: якщо встановити в 0 (False), елемент стане невидимим на формі	Усі видимі

Таблиця 2.4

Основні методи стандартних елементів керування

Метод	Пояснення	Для яких елементів застосовується
Clear()	Очищення вікна	textBox, richTextBox та інші
Hide()	Приховання елемента	Більшість видимих
Show()	Показ прихованого елемента	Більшість видимих

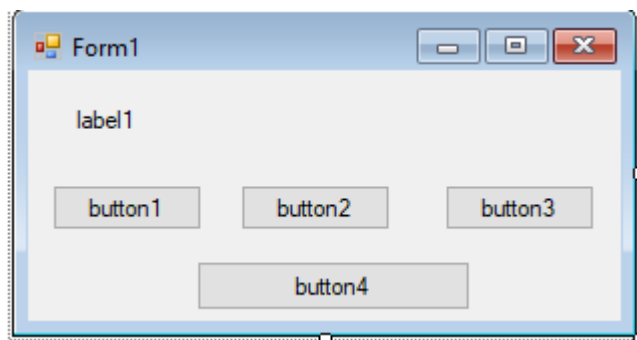
Приклади створення програмних проектів з інтерактивними формами

Приклад 2.1 Створити програму, в якій можна буде встановлювати колір форми натисканням кнопки. Слід передбачити можливість встановлення чотирьох кольорів: жовтого, синього, зеленого, червоного.

Розв'язок. На вікно форми слід розмістити такі елементи керування: один надпис (label) і чотири кнопки (button) (остання кнопка для виходу з програми). Ці елементи розташовані на панелі елементів керування. Для розміщення кожного елемента на форму треба:

- 1) на панелі елементів керування вибрати потрібний елемент (клацанням лівої кнопки по його піктограмі);
- 2) клацнути лівою кнопкою миші у тому місці форми, де елемент має міститися;
- 3) за потреби відкоригувати більш точно місце розташування та розміри елементів керування і самої форми. Це можна здійснити чи то за допомогою миші, чи задавши точні значення властивостей елементів у вікні властивостей.

Після того як ці дії повторити для розміщення всіх потрібних елементів, форма набуде вигляду



Примітка. Зауважимо, що за потреби розміщення на формі декількох елементів одного типу, є зручна можливість зробити це швидко, скористувавшись таким методом. На панелі елементів керування вибрати потрібний еле-

мент та натиснути клавішу [Ctrl]. Далі при кожному клацанні всередині форми, на ній з'являтиметься новий елемент.

Надписи, розмір, місце розташування на формі – все це властивості елементів керування, які можна задавати за допомогою вікна властивостей. Як вже зазначалось, щоб змінити властивості елемента, його треба виокремити, клацнувши по елементові лівою кнопкою миші, потім перейти до вікна властивостей, вибрати потрібну властивість і задати її нове значення. Отже, задамо властивості елементів згідно з таблицею, наведеною нижче.

Ім'я елемента	Властивість	Значення властивості	Пояснення
button1	Text	Червоний	Надписи на кнопках
button2	Text	Жовтий	
button3	Text	Зелений	
button4	Text	Вихід	Надпис на кнопці
	Size	170; 40	Ширина та висота кнопки
label1	TextAlign	MiddleCenter	Вирівнювання тексту
	AutoSize	False	Автоматичне змінювання розміру елемента заборонено
	Text	"Змінення кольору форми"	Текст надпису
	BackColor	Black	Колір фону елемента керування
	ForeColor	White	Колір шрифту надпису
	Font->Size	14	Розмір шрифту надпису
	Height	50	Висота елемента керування
	Width	300	Ширина елемента керування
Form1	Text	Приклад 2.1	Надпис форми

Після створення форми можна перейти до програмування. Колір форми має змінюватись при клацанні по відповідних кнопках. Для цього треба створити програми відгуків на подію Click для відповідних кнопок подвійним клацанням по цих кнопках. Після цього буде автоматично створено шаблон-заготовку в редакторі коду. Всі команди, які мають виконуватись при клацанні по кнопці, треба записати поміж операторних дужок { }. Програмний відгук на подію Click для кнопки button1 остаточно матиме вигляд:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    Form1::BackColor = Color::Red ;
}
```

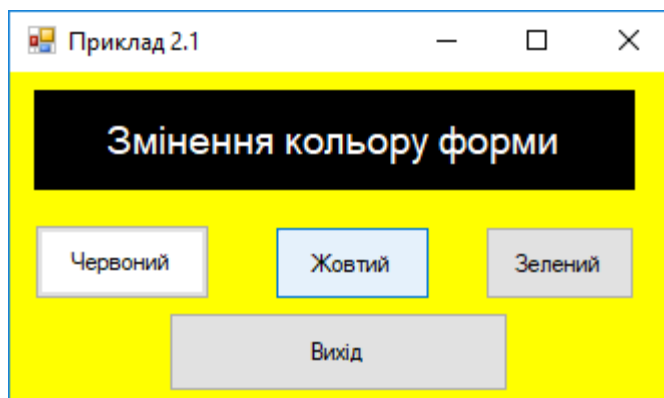
Програмний код для кнопок button2 та button3 відрізнятиметься від вищенаведеного лише кольором. Жовтий колір задається як Yellow, зелений – Green.

Для кнопки button4 (Вихід) програмний код міститиме команду закриття активної форми Close():

```
private: System::Void button4_Click(System::Object^ sender, System::EventArgs^ e)
{
    Close();
}
```

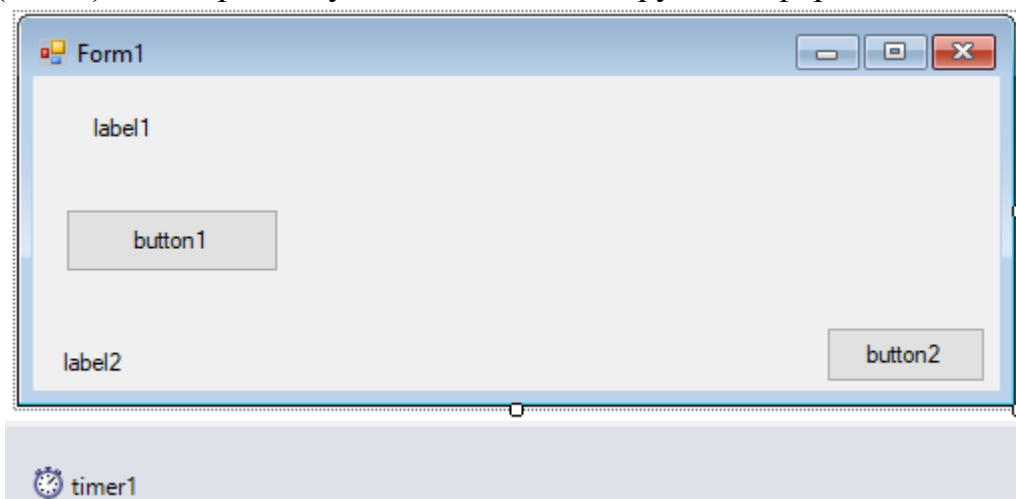
Перед запуском на виконання програмний проект доречно зберегти командою *File/SaveAll* (чи то через ярлик , чи клавішами [Shift] + [Ctrl] + [S]).

Для запуску проекту слід клацнути кнопку  чи то натиснути клавішу [F5]. Унаслідок цього має з'явитися наведена нижче форма проекту (це означає, що програму написано без помилок).



Приклад 2.2 Створити програму, в якій зреалізувати кнопку-«втікача», тобто кнопку, яка «втікатиме» за намагання натиснути її, окрім того, вивести поточну дату і час на форму.

Розв'язок. Розташувати на формі два надписи (label), дві кнопки (button) і таймер (timer). Після розташування елементів керування форма матиме вигляд:



Задамо властивості елементів керування, наведені в таблиці.

Ім'я елемента	Властивість	Значення властивості	Пояснення
Form1	Text	Приклад 2.2	Надпис на формі
	Size	550; 210	Ширина та висота форми
label1	AutoSize	false	Автоматичне змінювання розміру заборонено
	Size	500; 50	Ширина та висота елемента
	Text	Спробуй натиснути кнопку	Текст надпису
	BackColor	Blue	Колір фону
	ForeColor	White	Колір шрифту надпису
	Font->Size	14	Розмір шрифту надпису
	TextAlign	MiddleCenter	Вирівнювання тексту
Label2	Text	-	Текст надпису
button1	Text	Натисни!	Надпис на кнопці
button2	Text	Вихід	Надпис на кнопці

У вікні властивостей на вкладці подій (*Events*) слід задати шаблон для відгуку на подію `MouseMove` (подія виникає за наведення на кнопку покажчика миші) для кнопки `button1`. Для цього треба такі кроки:

- 1) виокремити елемент;
- 2) перейти до вікна властивостей;
- 3) відкрити вкладку подій;
- 4) двічі клацнути лівою кнопкою миші напроти назви потрібної події.

Після виконання наведених чотирьох кроків Visual C++ автоматично створить шаблон-заготовку в редакторі коду. Після того як вписати поміж операторних дужок такий програмний код, програма відгуку набуде вигляду:

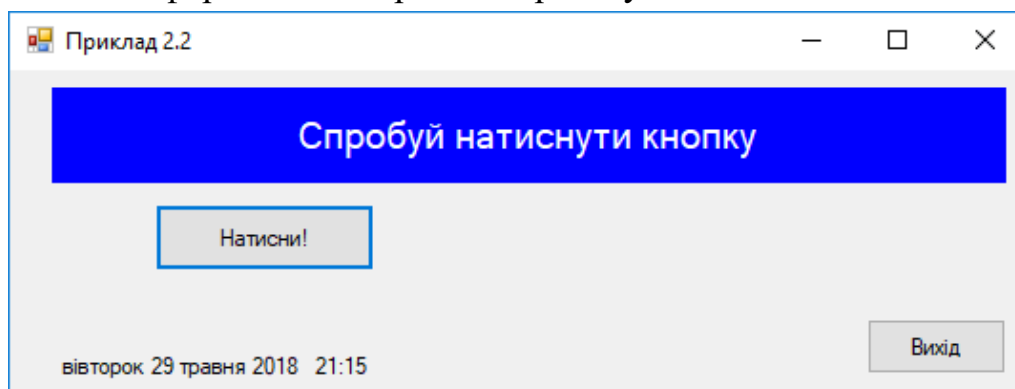
```
private: System::Void button1_MouseMove(System::Object^ sender,
                                         System::Windows::Forms::EventArgs^ e)
{
    if(button1->Left <= 0) button1->Left = Form1::Width - button1->Width - 20;
    else button1->Left -= 10;
}
```

Отже, кнопка «втікає», залишилось вивести дату й час на панель. Задамо для елемента `timer1` шаблон функції події `Tick` (подія відбувається із заданим у властивості `Interval` інтервалом часу), в якому запишемо програмний код:

```
private: System::Void timer1_Tick(System::Object^ sender, System::EventArgs^ e)
{
    label2->Text = DateTime::Now.ToString(L"dddd d MMMM yyyy HH:mm");
}
```

Пояснення. Функція `Now()` повертає поточні дату й час. Метод `ToString()` перетворює дату і час на текст у заданому форматі. Докладніше про це розглянуто в розд. 11.

Перед запуском на виконання програмний проект слід зберегти командою *File / SaveAll* (чи то через «гарячу» кнопку , чи клавішами `[Shift]+[Ctrl]+[S]`). Для запуску проекту на виконання слід натиснути клавішу `[F5]` чи «гарячу» кнопку . Вікно форми під час роботи проекту матиме вигляд:

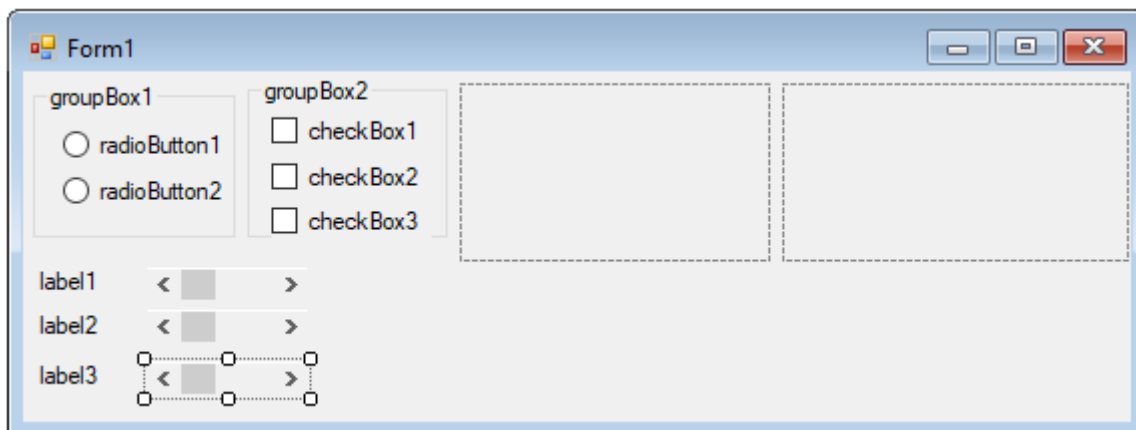


Приклад 2.3 Створити програму, в якій, залежно від вибору користувача, можна встановлювати колір першої чи другої панелі. Здійснити можливість змішувати кольори: задавати складові трьох основних кольорів: червоного, зе-

леного, синього. Надати можливість вибору дозволу/заборони змінювання певної складової кольорів.

Розв'язок. На формі слід розмістити такі елементи: три надписи (label), два елементи groupBox, дві панелі (panel), три смуги прокручування (hScrollBar). На елемент GroupBox1 розмістимо два елементи radioButton, а на GroupBox2 – три елементи checkBox.

Після розташування всіх елементів форма матиме вигляд:



Задамо властивості елементів згідно з таблицею, поданої нижче.

Елемент	Властивість	Значення властивості	Пояснення
Form1	Text	Приклад 2.3	Надпис на формі
label1	Text	R	Текст надпису
label2	Text	G	Текст надпису
label3	Text	B	Текст надпису
groupBox1	Text	Вибір панелі	Надпис елемента
	Size	90;90	Розміри елемента
groupBox2	Text	Дозволити складову кольору	Надпис елемента
	Size	175;90	Розміри елемента
radioButton1	Text	Панель № 1	Надпис елемента
	Checked	true	Стан вибору
radioButton2	Text	Панель № 2	Надпис елемента
hScrollBar1, hScrollBar2, hScrollBar3	Maximum	255	Верхня межа прокручування
	Width	550	Ширина смуги
checkBox1	Text	Червоний	Надпис елемента
	Checked	true	Стан вибору
checkBox2	Text	Зелений	Надпис елемента
	Checked	true	Стан вибору
checkBox3	Text	Синій	Надпис елемента
	Checked	true	Стан вибору

При змінюванні позиції повзунка смуги прокручування виникатиме подія Scroll. У нашому прикладі треба, щоб у відгук на цю подію комп'ютер змінював колір панелі panel відповідно до вибору перемикача radioButton. Новий колір визначається всіма трьома смугами прокручування: позиція повзунка кожної впливає на кількість відповідної складової частини кольору у результуючо-

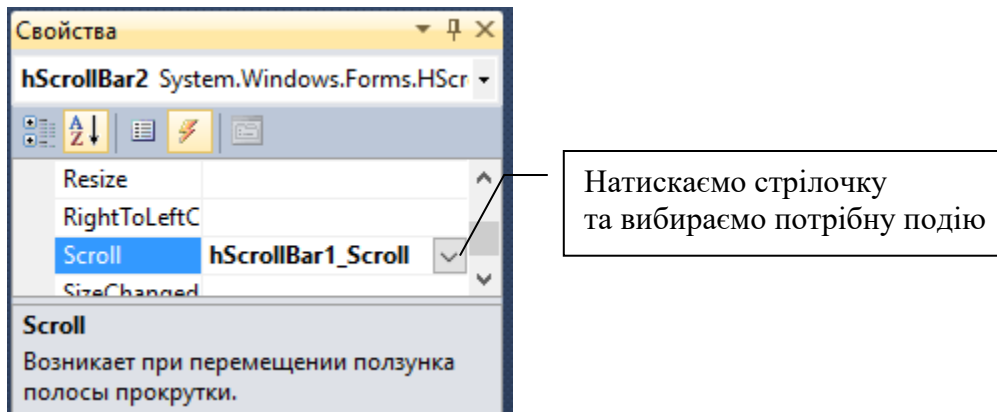
му кольорі при змішуванні. Для змішування трьох базових кольорів (Red-Green-Blue) використовується функція `FromArgb`, параметрами якої є позиції повзунків кожного з трьох кольорів.

Задамо шаблон функції для відгуку на подію `Scroll` для смуги прокручування `hScrollBar1`. Коли виконаємо чотири кроки, наведені у прикладі 2.2, матимемо шаблон у редакторі коду. Впишемо поміж операторних дужок такий програмний код:

```
private: System::Void hScrollBar1_Scroll(System::Object^ sender,
                                         System::Windows::Forms::ScrollEventArgs^ e)
{
    if (radioButton1->Checked) panel1->BackColor = Color::FromArgb(
        hScrollBar1->Value, hScrollBar2->Value, hScrollBar3->Value);
    else panel2->BackColor = Color::FromArgb(hScrollBar1->Value,
        hScrollBar2->Value, hScrollBar3->Value);
}
```

Програми для відгуку на подію `Scroll` для елементів `hScrollBar1` та `hScrollBar2` є аналогічні, але щоб не писати зайвого коду, зазначимо для `hScrollBar2` вже наявну програму-відгук `hScrollBar1_Scroll`. Для цього:

- ✓ виокремимо елемент `hScrollBar2`;
- ✓ у вікні властивостей перейдемо до вкладки подій;
- ✓ вибираємо для події `Scroll` зі списку `hScrollBar1_Scroll`:



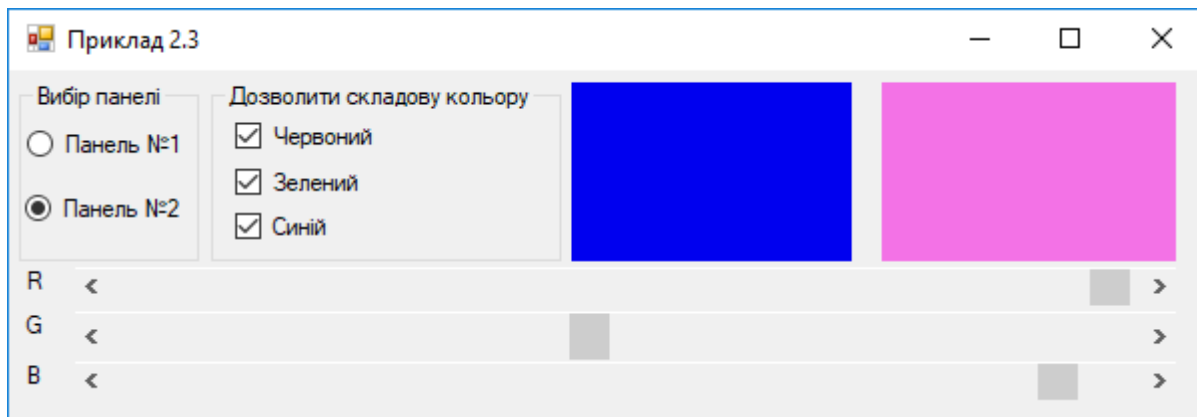
Аналогічно встановимо для смуги прокручування `hScrollBar3` програму-відгук для події `Scroll`.

Щоб дозволити чи заборонити змінення будь-якої складової кольорів, на формі є три відповідні «прапорці» (`CheckBox`). Якщо «прапорець» активовано (властивість `Checked`), то змінювати складову кольору можна, в іншому разі – ні. Стан «прапорця» змінюється за його натискання. Отже, для елементів `checkBox1`, `checkBox2`, `checkBox3` створимо програми опрацювання події `Click`, в яких запишемо:

```
для checkBox1: hScrollBar1->Enabled = checkBox1->Checked;
для checkBox2: hScrollBar2->Enabled = checkBox2->Checked;
для checkBox3: hScrollBar3->Enabled = checkBox3->Checked;
```

Після написання коду програми слід зберегти (*File / SaveAll* ) і запустити ([F5]) проект.

Вигляд форми під час роботи проекту:



2.2 Введення-виведення даних

Visual C++ підтримує декілька способів введення-виведення даних залежно від типу програмного проекту (консольний застосунок чи проект Windows Forms) та використовуваних команд.

2.2.1 Введення-виведення у стилі C++

Для введення-виведення даних у C++ найчастіше використовуються потокові команди **cin** (вхідний потік) та **cout** (вихідний потік) бібліотеки `iostream.h`, наприклад:

```
cout<< "Введіть число: ";
cin>> x;
cout<< "Квадрат цього числа: " << x*x << endl;
```

Перша з команда `cout` виведе рядок символів, який записується у подвійних лапках, і залишить курсор на тому ж самому рядку. При цьому символи рядка виводитимуться по одному, доки не зустрінеться завершальний символ `'\0'`. Друга команда дозволить ввести значення змінної `x`. Третя команда сформує рядок, поряд виведе числове значення та перемістить курсор на новий рядок (маніпулятор `endl`).

Наступна програма обчислення суми двох чисел наводилась як приклад на с. 18. У цій програмі є і команда потокового введення `cin`, і команди виведення `cout`:

```
1 | #include <iostream>
2 | using namespace std;
3 | int main()
4 | {
5 |     setlocale(0, ".1251");
6 |     double a,b;
7 |     cout << "Введіть два числа (через пробіл)\n";
8 |     cin >> a >> b;
9 |     cout << "Сума цих чисел - " << a + b << endl;
10 |     system ("pause>>void");
11 |     return 0;
12 | }
```

Результати виконання програми:

Введіть два числа (через пробіл)

5 10.2

Сума цих чисел – 15.2

Розглянемо детальніше наведену програму.

У рядку 1 директива `#include <iostream>` долучає файл `iostream` до коду програми. Це потрібно для використання команд `cin` та `cout` та пов'язаних з ним функцій. До речі, якщо в програмі буде використовуватись функція `getch()` для очікування натискання клавіші, слід на початку програми написати: `#include <conio.h>`, а для можливості використання функцій введення-виведення `scanf()` та `printf()` слід залучити директиву `#include <stdio.h>`.

Рядок 2 містить директиву `using` для простору імен (namespace) `std`, в якому доступу до команд `cin` та `cout` без застосування операції доступу (`::`). Якщо не записати цю директиву, то команди `cin` та `cout` слід буде записувати як `std::cin` та `std::cout`.

Рядок 3 містить заголовок функції `main()`. Функція `main()` є присутня у кожній програмі, саме до неї передається керування після завантаження та ініціалізації програми. Доволі часто у консолі функція `main()` не має аргументів, однак, якщо треба при викликанні програми передати їй якісь параметри, синтаксис функції матиме вигляд:

```
int main(array<System::String ^> ^args)
```

Тут аргумент `args` є вказівником на масив рядків, які містять аргументи. Усталено у консольній програмі Visual C++ функція `main()` має саме такий синтаксис. Для наведених у підручнику прикладів консольних програм синтаксис цієї функції є несуттєвим, тобто обидві форми написання є працездатними.

У рядку 5 застосовано команду `setlocale(0, ".1251")` для коректного відображення літер кирилиці при виведенні тексту. Цю команду можна записати і так: `setlocale(LC_ALL, "Ukrainian")`. Виведення замість літер кирилиці усілякої абракадабри спричинено тим, що Visual Studio у консольних програмах використовує для набраного тексту кодування Windows 1251, а для введенного тексту – кодування DOS. Коректно відобразити введенний за допомогою `cin` текст дозволить команда `setlocale(LC_ALL, ".OCP")`, повернувши початкові налаштування кодування.

Крім того, встановити потрібну кодову таблицю на потоки введення та виведення можна і такими способами:

```
system("chcp 1251");
```

або `system("chcp 1251 > null");`

або `#include <Windows.h>`

```
SetConsoleCP(1251);
```

```
SetConsoleOutputCP(1251);
```

Після цього у властивостях консольного вікна на вкладці *Шрифт* слід вибрати `Lucida Console`.

У рядку 6 оголошено дві змінні дійсного типу.

У рядку 7 показано найпростіший спосіб виведення на екран рядка інформаційного повідомлення командою `cout`. Символ `\n` є спеціальним форматним знаком. Він вказує `cout` перейти на новий рядок (аналогічно до `endl`).

Рядок 8 вводить два числа по черзі та записує їх у змінні `a` та `b`.

Рядок 9 ілюструє можливість виконання обчислення суми при виведенні за допомогою `cout`.

У рядку 10 виконується команда `system("pause>>void")` для того, щоб зупинити консольне вікно з результатами виконання програми. Якщо цього не зробити, то після виконання команд програми чорне вікно консолі закриється. Зазвичай це трапляється настільки швидко, що користувач не встигає побачити результати виконання програми. Щоб зупинити програму і надати користувачу можливість прочитати результати, використовується функція наведена команда наприкінці програми перед `return 0`. Аналогічну дію виконує функція `getch()` або `cin.get()`, які очікують на натискання будь-якої клавіші.

До речі, доволі зручною є можливість виведення за допомогою `cout<<` чисел не лише в десятковому форматі, а і в шістнадцятковому чи вісімковому, використовуючи модифікатори `dec`, `hex` і `oct` у середині вихідного потоку:

```
setlocale(0, ".1251");
cout<< "Вісімковий:\t\t " << oct << 10 << " " << 255 << endl;
cout<< "Шістнадцятковий:\t"<< uppercase<<hex<<10<<" " <<255<<endl;
cout<< "Десятковий:\t\t " << dec << 10 << " " << 255 << endl;
```

Результатом виконання цих команд будуть рядки:

```
Вісімковий:      12 377
Шістнадцятковий: A  FF
Десятковий:      10 255
```

Слід зазначити, що використання одного з цих модифікаторів залишиться в силі, допоки чи то програма не завершиться, чи то не буде використано інший модифікатор.

Для форматування даних при виведенні командою `cout<<` можна використовувати модифікатор `setw`, який дозволяє задавати ширину (мінімальну кількість символічних позицій) кожного виведеного числа. При цьому зазначена модифікатором ширина є дійсною лише для одного числа. Для можливості використання модифікаторів слід долучити заголовний файл `iomanip`. Якщо виведене число має меншу, аніж зазначену у модифікаторі `setw` ширину, перед ним будуть виведені пробіли. Так, команди

```
#include <iomanip>
. . . . .
cout<< "x=" << setw(1) << 155 << endl;
cout<< "x=" << setw(3) << 155 << endl;
cout<< "x=" << setw(5) << 155 << endl;
```

сформують рядки у такому вигляді:

```
x=155
x=155
x= 155
```

Тобто в останньому рядку перед значенням 155 виведуться два пробіли, а все число займе п'ять позицій.

При виведенні дійсних чисел, особливо коли йдеться про виведення матриць дійсних чисел, доречним є обмеження кількості знаків після десяткової крапки за допомогою маніпулятора **setprecision(int count)**:

```
cout << fixed << setprecision(3) << (13.5 / 2) << endl; // 6.750
cout << fixed << setprecision(2) << 24.16425 << endl; // 24.16
```

Використаний тут маніпулятор **fixed** задає виведення дійсних чисел з рухомою крапкою у форматі з фіксованою крапкою.

Команда **cout<<** дозволяє перенаправляти виведення на пристрій чи до файлу за допомогою операторів перепризначення виведення операційної системи. Однак повідомлення про помилки здебільшого недоречно спрямовувати до файлу і при цьому не виводити на екран. Для уникнення таких ситуацій існує спеціальний вихідний потік **cerr**, пов'язаний зі стандартним пристроєм помилок. Наприклад, команда

```
cerr<< "Повідомлення про помилку" << endl;
```

сформує відповідне повідомлення на екрані, а операційна система не дозволить перенаправити виведення на інший пристрій чи до файлу.

2.2.2 Введення-виведення у стилі C

1) Для форматowanego введення-виведення в C існують функції **scanf** (форматоване введення даних) та **printf** (форматоване виведення даних), які містяться в бібліотеці **stdio.h**. Для того щоб програма могла використовувати згадані команди, на початку програми слід долучити директиву **#include <stdio.h>**.

Обидві функції мають схожий формат:

```
scanf(<формат>, <список_змінних>);
printf(<формат>, <список_змінних>);
```

де: *формат* – рядок специфікаторів формату у подвійних лапках. Найбільш поширеними специфікаторами є: **%i** – для цілих чисел, **%f** – для дійсних чисел типу **float** чи **double**, **%s** – для рядка символів;

список_змінних – послідовність розділених комами змінних, значення яких вводиться чи виводяться.

Наприклад, функція **printf("x= %7.3f\n", x)** виведе дійсне число **x** у заданому форматі з трьома знаками дійсної частини числа після десяткової крапки.

При виконуванні функції **scanf()** відбувається таке. Програма призупиняє роботу і очікує, доки користувач набере на клавіатурі рядок символів та натисне клавішу [Enter]. Після натискання клавіші [Enter] функція **scanf()** перетворює введений рядок відповідно до специфікаторів формату на дані й записує їх до змінних, адреси яких зазначено. Наприклад, після виконання функції **scanf("%f", &vart)** і введення з клавіатури рядка 34.76 змінна **vart** набуде значення 34.76.

Функція **scanf("%i %f", &count, &vart)** введе значення цілої змінної **count** і дійсної змінної **vart**. Тут знак **&** є елементом синтаксису і означає операцію отримання адреси змінної у пам'яті. Докладніше про адреси змінних див. у розд. 6.

Специфікатор формату задає вигляд виведеного результату. У табл. 2.5 наведено найбільш поширені специфікатори формату. Необов'язковий параметр

специфікатора *n*, замість якого слід підставити десяткове число, задає розмір поля виведення; параметр *m* – розмір поля для виведення цифр дійсної частини.

Таблиця 2.5

Специфікатори формату

Специфікатор	Тип змінної	Формат виведення
%nd	<code>int</code>	десяткове зі знаком
%n.mf	<code>float</code> чи <code>double</code>	дійсне з фіксованою крапкою
%ne	<code>float</code> чи <code>double</code>	дійсне з рухомою крапкою
%nc	<code>char</code>	символ
%ns	<code>char*</code>	рядок

2) Існує ще одна пара функцій C для введення-виведення – `gets()` і `puts()`. Використовуються ці функції лише для рядків. Вони мають лише один аргумент – текст повідомлення для функції `puts` і змінну-рядок для функції `puts`, наприклад:

```
puts("Hello, Dolly");
char s[13];
gets(s);
```

Функцію `puts()` часто використовують для виведення на екран повідомлень. Вона після виведення автоматично переводить курсор на початок наступного рядка. Наприклад, функція

```
puts("Введіть значення, \n змінної x ");
```

виведе два рядки:

```
Введіть значення
змінної x
```

і переведе курсор на початок наступного рядка. Для обмеження рядків застосовується спеціальна послідовність символів.

Функція `gets(s)` зчитує символи з клавіатури до появи символу переведення рядка *Enter* і записує їх у рядок *s* (власне символ *Enter* до рядка не долучається, а замість нього записується нуль-символ).

Докладніше про введення-виведення рядків див. у розд. 7.

2.2.3 Введення-виведення у стилі .NET Framework

1) Клас **Console** є частиною бібліотек CLR (Common Language Runtime – середовище виконання .NET-застосунків) платформи .NET Framework і містить низку методів для введення-виведення потоків даних у консольних програмах. Цей клас належить до простору імен *System*, і тому слід записати команду:

```
using namespace System;
```

Методи цього простору `Read` і `ReadLine` призначені для читання символів зі стандартного потоку введення, а методи `Write` і `WriteLine` – для виведення рядка символів у стандартний потік виведення.

Специфікатори форматів:

f, *F* – формат з фіксованою дійсною крапкою;
d, *D* – формат *Decimal* (десяткові цілі числа);
c, *C* – грошовий формат;
r, *R* – відсотковий формат;

e, E – експоненційне представлення дійсних чисел з рухомою крапкою;
 g, G – формат General для перетворювання значень типу double до формату з фіксованою чи рухомою крапкою;
 n, N – формат Number для перетворення у форму ddd,ddd.dd

Приклади:

```
double x=54.321, y=0.0125; int n=1234;
Console.WriteLine();
Console.WriteLine("n= {0:d}", n );           // n= 1234
Console.WriteLine("n= {0:d5}", n );          // n= 01234
Console.WriteLine("x={0:f} y={1:f2}", x, y); // x=54.32   y=0.01
Console.WriteLine("Ставка: \t{0:p}", y );    // Ставка:   1.25 %
Console.WriteLine("Ціна товару: \t{0,7:c}",x); // Ціна товару:  $54.32
Console.WriteLine("y={0:e} x={0:E2}",y,x);   // y=1.250000e-002  x=5.43E+001
Console.WriteLine("n= {0:n}", n );           // n= 1,234.00
```

Крім цих специфікаторів форматів, можна застосовувати комплексні шаблони, наприклад:

```
Console.WriteLine("x={0:0.##}", x );           // x=54.32
Console.WriteLine("Ставка: \t{0:%#.##}", y ); // Ставка:      % 1.25
Console.WriteLine("y={0:#.##e+0},y);         // y=12.5e-3
```

2) Клас **Convert** простору імен System перетворює значення одного базового типу даних до іншого базового типу даних за допомогою того чи іншого методу перетворення. Наприклад, для перетворення до типу **double** існує метод **.ToDouble**, до типу **float** – метод **ToSingle**, до типу **int** – метод **ToInt32** тощо.

```
double x = Convert.ToDouble(textBox1->Text);
textBox2->Text = Convert.ToString(x*x);
```

2.2.4 Поширені функції перетворення числових типів даних в .NET

Введення і виведення даних у програмних проектах відбувається у текстовому вигляді. У середовищі .NET існує ціла низка методів для перетворення одних типів даних на інші.

Тип	Метод	Виконуване перетворення
String	ToDouble()	перетворює тип String на тип double
String	ToInt32()	перетворює тип String на тип int
double	ToString()	перетворює тип double на тип String
double	Parse()	перетворює тип String на тип double
int	ToString()	перетворює тип int на тип String
int	Parse()	перетворює тип String на тип int

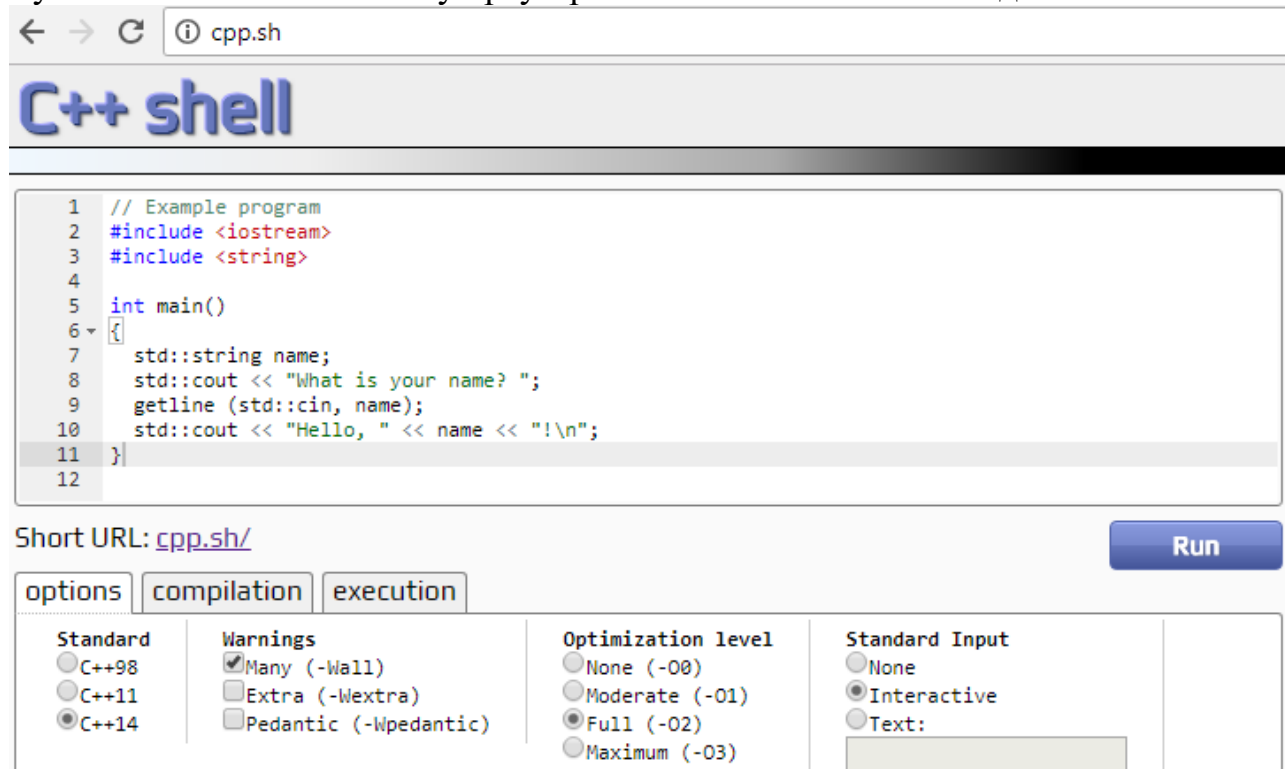
Наприклад:

```
int n = 3; double x;
// Виведення рядка з перетвореним цілим числом n
Console.WriteLine(n.ToString());
String^ s = L"3.14"; // Рядок символів
x = Double.Parse(s); // Перетворення рядка s на дійсне число x
```


2.3 Робота в онлайн-компіляторах мовою C++

2.3.1 C++ shell

C++ shell (<http://cpp.sh>) – онлайн-компілятор мовою C++. При початковому завантаженні C++ shell у браузері його вікно матиме вигляд:



```
1 // Example program
2 #include <iostream>
3 #include <string>
4
5 int main()
6 {
7     std::string name;
8     std::cout << "What is your name? ";
9     getline (std::cin, name);
10    std::cout << "Hello, " << name << "!\n";
11 }
12
```

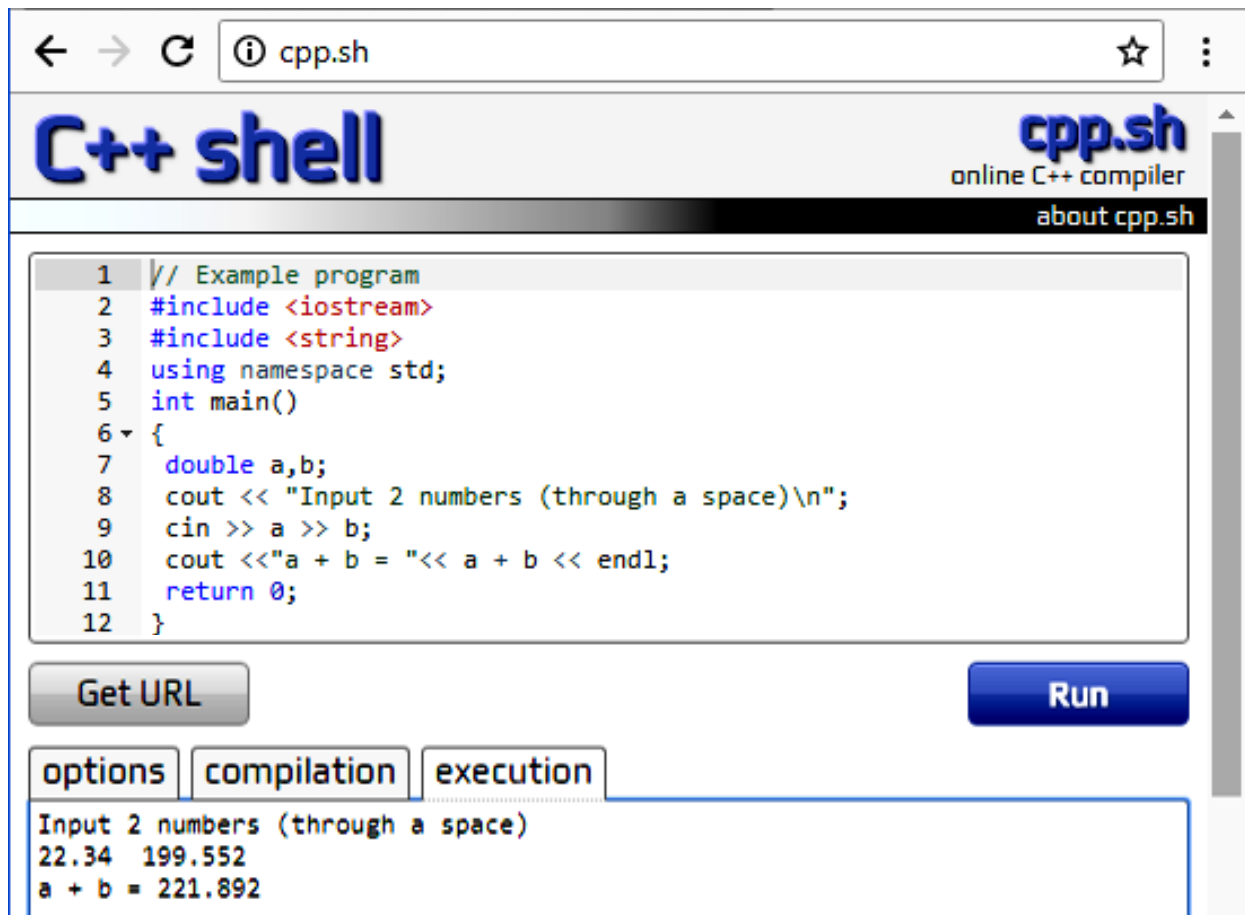
Short URL: cpp.sh/ Run

options compilation execution

Standard ☐ C++98 ☐ C++11 ☒ C++14	Warnings ☒ Many (-Wall) ☐ Extra (-Wextra) ☐ Pedantic (-Wpedantic)	Optimization level ☐ None (-O0) ☐ Moderate (-O1) ☒ Full (-O2) ☐ Maximum (-O3)	Standard Input ☐ None ☒ Interactive ☐ Text:

Онлайн-компілятор C++ shell має суттєві переваги: простий короткий URL, використання стандартних для C++ команд інтерактивного введення-виведення, проста можливість поділитися кодом, скориставшись кнопкою *Get URL*, різні рівні оптимізації (O0, O1, O2, O3), вибір стандартів C++ (C++98, C++11, C++14), підтримка попереджень (Wall, Wextra, Wpedantic) тощо. До недоліків можна віднести: роботу тільки з однією мовою програмування C++, можливість транслювати лише однофайлові проекти, відсутність підтримки програмування файлів, періодичні відключення та зависання, відсутність підтримки кирилиці.

Наявний у вікні програмний код слід замінити на власний. Наприклад, розглянутий на с. 18 програмний код для обчислення суми двох введених чисел разом з результатами (внизу вікна) можуть мати вигляд:



2.3.2 Ideone

Ideone (<https://ideone.com>) – це онлайн-інструмент для компілювання та налагодження програмного коду більш ніж 60-ти мовами програмування, у тому числі мовою C++. Ideone підтримує кирилицю, дозволяє поширювати (ділитися) кодом, завантажувати код у вигляді файлу з розширенням cpp. Можна записувати анотації до коду в окремому вікні. Проте, не можна задавати параметри компіляції та передавати параметри командного рядка.

При початковому завантаженні <https://ideone.com> у браузері його вікно з вибором мови програмування матиме вигляд:

Надійне | <https://ideone.com>

ideone.com

</> enter your source code or insert [template](#) or [sample](#)

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
```

popular others Bash (bash 4.4.5)

Bash	Pascal	Ada95	Common Lisp	Java	Prolog
C	Perl	Assembler 32b	Common Lisp	JavaScript	Python
C#	PHP	Assembler 32b D		JavaScript	Python 3 nbc
C++	Python	Assembler 64b D		Kotlin	R
C++14	Python 3	AWK	D	Lua	Racket
Haskell	Ruby	AWK	Dart	Nemerle	Rust
Java	SQLite	BC	Elixir	Nice	Scala
Objective-C	Swift	Brainf**k	Erlang	Nim	Scheme
Pascal	VB.net	C	F#	Node.js	Scheme
		C++ 4.3.2	Fantom	Objective-C	Scheme
		C++14	Forth	Ocaml	Smalltalk
		C99	Fortran	Octave	TCL
		Clips	Go	Perl	Text
		Clojure	Gosu	Pico Lisp	Unlambda
		Cobol	Groovy	Pike	VB.NET
		COBOL 85	Icon	Prolog	Whitespace
		CoffeeScript	Intercal		

Наведений у п. 2.3.1 програмний код програмний код для обчислення суми двох введених чисел разом з результатами в Ideone матиме вигляд:

Надійне | <https://ideone.com>

</> enter your source code or insert [template](#) or [sample](#)

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     double a,b;
6     cout<<"Input 2 numbers\n";
7     cin>>a>>b;
8     cout<<"a+b="<<a+b;
9     return 0;
10 }
```

input Output ☒ syntax highlight

22.55 45.88

save ideone it!

stdout

Input 2 numbers
a+b=68.43

2.3.3 Codepad

Codepad (<http://codepad.org>) – простий онлайн-компілятор / інтерпретатор різними мовами. Після компіляції вставленого програмного коду буде сформовано URL-адресу, якою можна поділитися в чаті або через електронну пошту, щоб отримати допомогу або щоб показати комусь власні наопрацювання.

Вигляд вікна Codepad у браузері з програмним кодом для обчислення суми двох введених чисел матиме вигляд:

codepad is an online compiler/interpreter, and a simple collaboration tool.
Paste your code below, and codepad will run it and give you a short URL you can use to share it in chat or email.

Language:

- ☐ C
- ☒ C++
- ☐ D
- ☐ Haskell
- ☐ Lua
- ☐ OCaml
- ☐ PHP
- ☐ Perl
- ☐ Plain Text
- ☐ Python
- ☐ Ruby
- ☐ Scheme
- ☐ Tcl

```
#include <iostream>
using namespace std;
int main()
{
    double a,b;
    cout<<"Input 2 numbers\n";
    cin>>a>>b;
    cout<<"a+b="<<a+b;
    return 0;
}
```

☐ Private ☒ Run code

2.3.4 Wandbox

Wandbox (<https://wandbox.org>) – онлайн-компілятор з потужним функціоналом, зручним вибором мови програмування та версії стандарту C++, підтримкою декількох файлів та різних редакторів клавіатурних скорочень.

Окрім стандартної бібліотеки, в ньому є декілька бібліотек, включаючи Boost (до 1.67), Sprout і MessagePack. Скомпільовані програми запускаються в середовищі пісочниці, що дозволяє вводити і виводити файли. Зручним є підтримка кирилиці та запуск на виконання відповідною кнопкою або клавішами [Ctrl]+[Enter]. Крім того, можна поширювати код через посилання.

Вигляд вікна Wandbox у браузері з програмним кодом для обчислення суми двох введених чисел матиме вигляд:

Надійне

<https://wandbox.org>

Load template

☒ Warnings
☐ Optimization
☐ Verbose

Boost 1.67.0 ▾

☐ Sprout
☐ MessagePack

C++2a(GNU) ▾

no pedantic ▾

Compiler options:

```

1 #include <iostream>
2 #include <cstdlib>
3 using namespace std;
4 int main()
5 {
6     double a,b;
7     cout<<"Введіть 2 числа\n";
8     cin>>a>>b;
9     cout<<"a+b="<<a+b;
10    return 0;
11 }

```

\$ g++ prog.cc -Wall -Wextra -I/opt/wandbox/

C++

Languages

Bash script 1	Lisp 5
C 41	Lua 8
C# 0	Nim 4
C++ 43	OCaml 5
CPP 2	OpenSSL 8
CoffeeScript 8	PHP 5
Crystal 0	Pascal 4
D 10	Perl 8
Elixir 8	Pony 3
Erlang 6	Python 18
F# 5	Rill 1
Go 8	Ruby 21
Groovy 3	Rust 3
Haskell 7	SQL 4
Java 8	Scala 6
JavaScript 13	Swift 8
Lazy K 1	Vim script 8

Stdin

42.58 10.8

Run (or Ctrl+Enter)

#3

#2

Share

Code

Start

Введіть 2 числа

a+b=53.38

Питання та завдання для самоконтролю

- 1) Пояснити поняття RAD, GUI та IDE.
- 2) Що таке .NET Framework?
- 3) У чому специфіка середовища CLR?
- 4) Назвати різні способи створення програмних проектів у Visual C++.
- 5) Що таке консольна програма?
- 6) Назвати послідовність дій щодо створення консольного проекту у Visual C++.
- 7) Перелічити засоби введення-виведення даних у стилі C++.
- 8) Перелічити засоби введення-виведення даних у стилі C.
- 9) У чому схожість і відмінність функцій `scanf()` та `printf()`?
- 10) Як виглядатиме функція `scanf()` для введення значення дійсної змінної `X`?
- 11) Як виглядатиме функція `scanf()` для введення значення цілої змінної `X`?
- 12) Перелічити засоби введення-виведення даних у стилі .NET Framework.
- 13) Які функції введення-виведення даних у консольному режимі існують?
- 14) Назвати функції форматowanego введення-виведення в C.
- 15) Назвати потокові команди введення-виведення даних у C++.
- 16) Яка послідовність створення консольного проекту CLR у Visual C++?
- 17) Назвати послідовність дій щодо створення програмного проекту з формою у Visual C++.
- 18) За допомогою якої директиви до програми долучають бібліотечні модулі (заголовні файли)?
- 19) Які заголовні файли треба долучити для використання функцій введення-виведення?
- 20) В яких бібліотеках містяться математичні функції?
- 21) Як можна у Visual C++ запустити проект на виконання?
- 22) В який спосіб можна зберегти програмний проект у Visual C++?
- 23) Назвати основне призначення сторінки властивостей.
- 24) Назвати різні події, які доступні для різних елементів керування.
- 25) Назвати основні властивості, притаманні елементам керування.
- 26) Вказати основні методи стандартних елементів керування.
- 27) Назвати функції перетворення числових типів даних в .NET.
- 28) Яке призначення мають фігурні та круглі дужки у C++?
- 29) Вказати призначення і можливий синтаксис функції `main()`.
- 30) Назвати та охарактеризувати онлайн-компілятори мовою C++.

РОЗДІЛ 3

ЕЛЕМЕНТИ МОВИ C++

3.1 Історія C++

Мова програмування – формалізована мова, призначена для описування алгоритмів розв’язування задач на ЕОМ. Мова програмування потрібна, щоб записувати алгоритм у термінах, зрозумілих компіляторів, який, своєю чергою, перекладе його на зрозумілу процесорові машинну мову. Як і кожна людська мова, мова програмування складається з алфавіту (вживаних символів), слів (елементарних конструкцій, які називають лексемами), словосполучень (виразів) та речень (операторів). Лексеми утворюються із символів, вирази – із лексем та символів, а оператори – із символів, виразів та лексем.

C++ є результатом еволюції мови C, створеної Денісом Рітчі та Брайаном Керніганом. Ще на початку 70-х років минулого сторіччя C набула широкої популярності як мова розробляння операційної системи UNIX. Сьогодні більшість операційних систем написано мовою C та/чи C++. Мова C є доступною для більшості систем, до того ж вона є машинонезалежною. За ретельного проектування можна писати C-програми, які можна переносити на більшість комп’ютерів.

Широке використання C на комп’ютерах різних апаратних платформ призвело до появи різноманітних варіацій мови. Це зумовило потребу стандартизації мови і 1990 року відповідний документ стандарту ANSI/ISO 9899: 1990 було розроблено.

Мову C++ розробив Бьєрн Страуструп на початку 80-х років минулого сторіччя. Порівняно із C, мова C++ набула багато «прикрас», головною з яких є можливість об’єктно-орієнтованого програмування. Об’єкти є за своєю суттю програмними компонентами, які моделюють предмети реального світу. Модульний, об’єктно-орієнтований підхід до проектування та реалізації програмного забезпечення дозволили полегшити написання складних потужних програмних проектів. Об’єктно-орієнтовані програми є набагато простіші для розуміння, виправлення та модифікації.

1991 року на основі C і C++ фірмою Sun, а саме Джеймсом Гослінгом, було створено мову Java. Метою Java на початку було керування інтелектуальною побутовою технікою, а згодом, завдячуючи шаленому вибухові популярності World Wide Web, її швидко було перекваліфіковано і представлено 1995 року як потужний засіб розробляння програм для розв’язування різноманітних сучасних задач. Сьогодні Java використовується для створювання динамічного й інтерактивного вмісту веб-сторінок, розширювання можливостей веб-серверів, створювання програм для мобільних телефонів та побутових пристроїв, розроблення широкомасштабних програмних проектів на підприємствах тощо. Подібні можливості має сучасна версія C++, як-от Microsoft® Visual C++®.NET. На основі C++ і Java для платформи .NET було розроблено мову C#,

яка, подібно до Microsoft® Visual C++®.NET та Visual Basic.NET, дозволяє інтегрувати в програмні проекти можливості Internet та Web, писати компоненти програмного забезпечення мовою, яку знає розроблювач, а потім будувати проекти, комбінуючи ці компоненти з компонентами, написаними будь-якою іншою .NET мовою.

Microsoft Visual C++ (MSVC) – інтегроване середовище розроблення програмного забезпечення мовою C++, розроблена фірмою Microsoft. Постачається або як частина комплекту Microsoft Visual Studio, або окремо у вигляді безкоштовного функціонально обмеженого комплекту Visual C++ Express Edition.

Visual C++.NET дозволяє генерувати код як для платформи .NET Framework, так і для виконання у середовищі «чистої» Windows. У цьому відношенні Visual C++.NET є унікальним серед інших мовних засобів, що надаються середовищем Visual Studio, оскільки ані Visual Basic .NET, ані Visual J# не здатні генерувати код для чистої Win32.

Отже, C++ – потужна, лаконічна, гнучка, мобільна і жива мова, яка розвивається і в різних своїх версіях набуває нових сучасних можливостей.

3.2 Алфавіт мови C++

Алфавіт мови – набір символів, які можна використовувати для записування програмного коду: це великі та малі літери латиниці, цифри, знаки операцій та спеціальні символи. Кожен зокрема чи в комбінаціях *знаки операцій* і *спеціальні символи* дозволяють задавати лексеми, вирази і оператори, серед яких найбільш використовувані є такі:

= – присвоювання. Наприклад, якщо x і y – змінні, то запис $x = y$ означає, що змінна x набуває значення y (докладніше див. п. 3.5.3);

+, −, *, / – знаки операцій додавання, віднімання, множення та ділення;

++, -- – знаки операцій збільшення та зменшення на одиницю значення операнда;

; – символ завершення команди;

< (менше), <= (менше чи дорівнює), > (більше), >= (більше чи дорівнює), == (дорівнює), != (не дорівнює) – знаки операцій порівняння (див. п. 4.3.3);

+=, -=, *=, /=, %=, <<=, >>=, ?=, ^=, |= – знаки для записування операцій присвоювання (див. п. 3.5.3);

% – залишок від цілочисельного ділення двох чисел. Наприклад, у виразі $c = a \% b$ значення залишку від ділення a на b присвоюється змінній c ;

&&, ||, ! – знаки логічних операцій, які докладно описано в п. 4.3.3;

&, |, ^, <<, >>, ~ – знаки побітових логічних операцій (див. п. 4.3.3);

?: – знаки умовної операції (див. п. 4.3.7);

*, & – знаки адресних операцій. Хоча такі символи вже зустрічались у цьому переліку, однак їхнє специфічне застосовування у відповідних виразах дозволяє позбавитись усякої неоднозначності (див. розд. 6);

/*, */, // – сукупність знаків, використовуваних для записування коментарів;

, – знак коми, використовуваний для перелічування об'єктів;
• , -> – знаки використовувані при роботі зі структурованими типами і класами;
{ } – операторні дужки, які поєднують блок команд;
[] – дужки для індексів масивів;
() – дужки для встановлення черговості виконання операцій і записування аргументів функції;
' ' – знаки для записування значення символьних констант;
" – знак для записування значення символьної інформації.

Список усіх операцій у порядку спадання їхніх пріоритетів наведено у додатку Б.

Коментарі застосовуються лише для пояснень і жодних дій у програмі не спричиняють, тому що ігноруються компілятором. Текст коментарів розміщують поміж знаків `/*` та `*/`. Коментар може бути будь-якої довжини, чи то займати частину рядка, чи декілька рядків. Наприклад:

```
/* Коментар до програми  
   може займати кілька рядків    */
```

Окрім символів `/*` та `*/` для невеликих коментарів в один рядок в C++ використовують знак `//`.

```
// Текст після цього символу і до кінця рядка є коментарем.
```

Також, окрім пояснень, доволі часто коментарі використовують при налагодженні для тимчасового «вимкнення» певного фрагмента програми.

Змінна – іменована ділянка оперативної пам'яті, застосовувана для зберігання даних під час роботи програми. При оголошенні змінної для неї резервується певна ділянка пам'яті, розмір якої залежить від конкретного типу змінної. *Значення змінної* – фактичне значення, яке міститься у цій ділянці пам'яті.

Ім'я (ідентифікатор) змінної може складатися з літер латиниці, цифр та символу «`_`» (підкреслювання), але неодмінно має розпочинатися з літери чи символу підкреслювання. Великі й малі літери розрізняються, тобто мова C++ є чутливою до регістру, наприклад – *a* та *A* – це два різних об'єкти. Ідентифікатор створюється на етапі оголошення змінної, функції, структури тощо. Після цього його можна використовувати в командах розроблюваної програми.

Слід відзначити важливі *особливості при вибиранні ідентифікатора*.

По-перше, ідентифікатор не повинен збігатися з ключовими словами, із зарезервованими словами й іменами функцій бібліотек компілятора мови C++.

По-друге, слід звернути особливу увагу на використання символу підкреслення «`_`» в якості першого символу ідентифікатора, позаяк побудовані у такий спосіб ідентифікатори, з одного боку, можуть збігатися з іменами системних функцій та змінних, а з іншого, – при використанні таких ідентифікаторів програми можуть стати непереносними, тобто їх не можна буде використовувати на комп'ютерах інших типів.

3.3 Типи даних

У програмі мовою C++ усі змінні мають бути оголошеними, тобто для кожної змінної має бути зазначено її тип. На відміну від інших мов, у C++ *задавати тип змінної можна в будь-якому місці програми до її використання*. При оголошенні змінної для неї резервується ділянка пам'яті, розмір якої залежить від типу змінної.

Тип змінної – вид і розмір даних, які змінна може зберігати. Кожен тип даних зберігається й опрацьовується за певними правилами. Слід зауважити, що розмір одного й того ж типу даних може відрізнятися на комп'ютерах різних платформ, а також залежить від налагоджень компілятора.

Усі типи мови C++ розподіляють на дві групи: основні типи та структуровані. Список основних типів даних C++ із зазначенням діапазону та прикладами можливих значень змінних наведено в табл. 3.1.

Таблиця 3.1

Основні типи даних C++

Тип	Назва	Розмір, байт	Діапазон	Приклади можливих значень	Типи чисел
<code>char</code>	символьний (знаковий)	1	–128...127	'a', '\n', '9'	цілі
<code>unsigned char</code>	беззнаковий символьний	1	0...255	1, 233	
<code>short</code>	короткий цілий	2	–32 768...32 767	1, 153, –349	
<code>unsigned short</code>	беззнаковий короткий	2	0...65 535	0, 4, 65 000	
<code>int</code>	цілий (знаковий)	4*	–2 147 483 648... ...2 147 483 647	–30 000, 0, 690	
<code>unsigned int</code>	беззнаковий цілий	4	0...4 294 967 295	2 348, 60 864	
<code>long</code>	цілий (знаковий)	4	–2 147 483 648... ...2 147 483 647	–30 000, 0, 690	дійсні
<code>float</code>	дійсний одинарної точності	4	$3.4 \cdot 10^{-38}$... $3.4 \cdot 10^{38}$	3.23, –0.2, 100.23, 12, –0.947, 0.0001, 6.34e–3, 4e5	
<code>double</code>	дійсний подвійної точності	8	$1.7 \cdot 10^{-308}$... $1.7 \cdot 10^{308}$		
<code>long double</code>	довгий дійсний	10	$3.4 \cdot 10^{-4932}$... $1.1 \cdot 10^{4932}$		
<code>bool</code>	логічний	1	false чи true	false(0), true(>=1)	
<code>enum</code>	перераховний	2 чи 4			
<code>void</code>	порожній, без значення				
* – залежно від налагоджень компілятора та апаратних характеристик тип <code>int</code> може мати 4 чи 2 байти.					

До основних (базових) типів можна віднести `char`, `int`, `float` та `double`, а також їхні варіанти із специфікаторами `short` (короткий), `long` (довгий), `signed` (зі знаком) та `unsigned` (без знаку).

Структуровані (похідні) типи базуються на основних, до них належать масиви будь-яких типів, вказівники, функції, класи, файли, структури, об'єднання, перерахування тощо.

Для подання *цілих чисел* використовують типи `char`, `short`, `int`, `long`. Специфікатор `unsigned` застосовують при роботі з додатними числами (без знаку), а специфікатор `signed` – для яких завгодно чисел, як додатних, так і від'ємних. За промовчанням призначається знаковий тип, а тому специфікатор `signed` зазначати необов'язково.

Типи `float` і `double` визначають *дійсні змінні* розміром у 32 і 64 біти відповідно, а `long double` – 80 бітів. В C++ для відокремлення цілої частини числа від дійсної застосовується десяткова крапка. Окрім звичної форми, дійсні константи можна записувати у формі з рухомою крапкою. Наприклад: $2.38e3$ (яке дорівнює $2.38 \cdot 10^3 = 2380$), $3.61e-4$ (яке дорівнює $3.61 \cdot 10^{-4} = 0.000361$). Число перед символом «e» називається *мантисою*, а після символу «e» – *порядком*, тобто замість основи 10 використовується літера «e», після якої ставиться показник степеня, наприклад: $1e3$ ($1 \cdot 10^3 = 1000$), $-2.7e-4$ ($-2.7 \cdot 10^{-4} = 0.00027$). Змінні типу `float` займають в пам'яті 4 байти: 1 біт – знак, 8 бітів – порядок, 23 біти – мантика.

Отже, на базових типах (`char`, `int`, `float`, `double`) будується решта типів даних за допомогою специфікаторів:

`signed` (знаковий – за промовчанням), `unsigned` (беззнаковий) – до цілих типів `char` і `int`;

`long` (довгий), `short` (короткий) – до типів `double` і `int`.

Зауважимо, що ключові слова `signed` і `unsigned` є необов'язковими. Вони визначають, як інтерпретується старший біт оголошеної змінної, тобто, якщо визначено тип `signed`, то нульовий біт числа виділяється під знак (0 – число додатне, 1 – від'ємне). При визначенні типу `unsigned` усі виділені біти разом зі старшим використовується для зберігання значення числа. Якщо специфікатор типу складається з ключового типу `signed` чи `unsigned` і далі одразу слідує ідентифікатор змінної, то вона розглядатиметься як змінна типу `int`.

Наприклад:

```
unsigned int n;  
int c;           // Інтерпретується як signed int c  
short a;        // Інтерпретується як signed short int a  
unsigned d;     // Інтерпретується як unsigned int d  
signed f;       // Інтерпретується як signed int f
```

При оголошенні змінних їм можна присвоювати початкові значення, які надалі може бути змінено. У прикладі

```
int i, j = 1, q = 0xFFFF;
```

ціла змінна `i` не є ініціалізована, `j` – ініціалізована значенням 1, `q` – ініціалізована шістнадцятковим значенням `0xFFFF` (десятькове 4095).

При ініціалізації змінних їм можна присвоювати арифметичні вирази:

```
long megabute=1024*1024;
```

Наведемо ще кілька прикладів оголошування змінних:

```
char tol='a'; // Символьна змінна tol ініціалізується символом 'a'
```

```
char x,c='\n'; // x не є ініціалізований, c – ініціалізований символом [Enter]
```

```
char st[6]="Одеса"; // Масив з 5-ти символів зі значенням "Одеса"
```

Якщо при оголошенні числові змінні не ініціалізовано, то їхні значення є невизначені (випадкові).

Зауважимо, що тип `char` використовується для подання символу, а для визначення його десяткового коду – тип `unsigned char` чи `signed char`. Значенням змінної типу `char` є відповідний код символу (розміром 1 байт). Для визначення десяткових кодів символів кирилиці специфікатор типу ідентифікатора даних має бути типом `unsigned char`. Окрім того, тип `char` застосовується для оголошення рядків типу `char*` (див. розд. 7).

Змінна логічного типу `bool` займає лише 1 байт і може набувати лише одне із двох значень: `true` (істина) чи `false` (хибність). Зазвичай змінні типу `bool` застосовують чи то як перемикачі чи для побудови логічних виразів в умовах. Слід зауважити, що в C++ змінним логічного типу можна присвоювати і цілочисельні значення; при цьому значення 0 розцінюється як `false`, а будь-яке ненульове значення – як `true`.

Перерахування `enum` (від *enumerate*) призначено для вибирання й призначання об'єктів із заданої множини значень. Оголошення перерахувань задає відповідний тип змінної і визначає список логічно пов'язаних іменованих констант, значеннями яких є певні цілі числа. Змінна перерахування типу може набувати значення однієї з іменованих констант списку. Кожен ідентифікатор у списку іменує елемент з переліку і має бути унікальним. Максимальна кількість ідентифікаторів – 65 535. Ідентифікатори списку є еквівалентні до їхніх значень, які можуть бути і додатними і від'ємними. У разі відсутності конкретного значення, першому ідентифікаторові відповідає значення 0, наступному ідентифікаторові – значення 1 тощо, наприклад:

```
enum days { sun, mon, tues, wed, thur, fri, sat } anyday;
```

В цьому прикладі оголошено перерахування `days` з відповідною множиною значень, а також змінну `anyday` типу `days`. Імена `sun`, `mon`, `tues`, `wed`, `thur`, `fri`, `sat` є цілими константами, перша з яких `sun` автоматично встановлюється в 0, а кожна наступна має значення на одиницю більше за попередню (`mon` = 1, `tues` = 2, ..., `sat` = 6).

Властивості даних типу `enum` є аналогічними до властивостей даних типу `int`, а змінні типу `enum` можуть застосовуватись в індексних виразах, в арифметичних операціях і в операціях порівняння у якості операндів. Більш докладно цей тип розглянуто в підрозд. 10.4.

Тип `void` (порожній, без значення) використовується, по-перше, для визначення типу функцій, які не повертають жодного результату, а всі дії та обчислення виконуються всередині цих функцій. По-друге, цей тип застосовують для зазначення порожнього списку аргументів функції (докладно функції роз-

глядаються в розд. 8). По-третє, `void` використовують як базовий тип для вказівників. Окрім того, його застосовують в операціях зведення типів.

Для визначення розміру пам'яті, займаної змінною, існує операція `sizeof()`, яка повертає значення довжини зазначеного типу, наприклад:

```
a = sizeof(int);           // a = 4
b = sizeof(long double);  // b = 10
```

Окрім основних типів C++, середовище Visual C++ додатково підтримує власні різновиди типів в просторі імен `System`, спільні для мов платформи .NET Framework (табл. 3.2).

Таблиця 3.2

Додаткові типи даних Visual C++ платформи .NET Framework

Тип Visual C++	Тип платформи .NET Framework
<code>bool</code>	<code>System.Boolean</code>
<code>signed char</code>	<code>System.SByte</code>
<code>unsigned char</code>	<code>System.Byte</code>
<code>wchar_t</code>	<code>System.Char</code>
<code>double</code> та <code>long double</code>	<code>System.Double</code>
<code>float</code>	<code>System.Single</code>
<code>int</code> , <code>signed int</code> , <code>long</code> та <code>signed long</code>	<code>System.Int32</code>
<code>unsigned int</code> та <code>unsigned long</code>	<code>System.UInt32</code>
<code>__int64</code> та <code>signed __int64</code>	<code>System.Int64</code>
<code>unsigned __int64</code>	<code>System.UInt64</code>
<code>short</code> та <code>signed short</code>	<code>System.Int16</code>

Крім зазначених в табл. 3.2, середовище виконання CLR підтримує рядки як об'єкти, для яких пам'ять виділяється автоматично. Оголошують рядки типу `String` за допомогою модифікатора дескриптор-об'єкта (^):

```
String^ s2 = L"Програмування розвиває мозок";
```

Оголошення типів. Окрім оголошень змінних різних типів, в C++ є можливість оголошувати власні типи. Це можна здійснювати у два способи. Перший спосіб – використовувати для оголошення типу ключове слово `typedef`. Другий спосіб (буде розглянуто докладніше в розд. 10) – зазначити ім'я тега при оголошенні структури, об'єднання чи перерахування, а потім застосовувати це ім'я при оголошенні змінних та функцій у якості посилання на цей тег.

При оголошенні ключовим словом `typedef` створюється новий тип даних (див. підрозд. 10.1), який надалі може бути використаний у межах видимості цього типу для оголошення змінних, наприклад:

```
typedef int vector[40];
vector x;           // Змінна x – масив 40-ка цілих чисел
```

Що є еквівалентним до оголошення: `int x [40];`

Використовуючи ключове слово `typedef`, можна оголошувати які завгодно типи, включаючи вказівники, функції чи масиви.

3.4 Константи

Константа — величина, яка не змінюється упродовж виконання програми. Для оголошення константи у програмі використовується специфікатор `const`. При цьому зазначається тип константи, наприклад:

```
const float Pi = 3.14159;
```

У якості значення константи можна подавати константний вираз, який містить раніш оголошені константи та змінні. Наприклад:

```
const float Pi2 = 2 * Pi, k = Pi / 180;
```

Константам при створенні слід неодмінно надавати значення, які в подальшому не можна буде змінювати.

У мові C++ розрізняють чотири типи констант: цілі, дійсні, символьні константи та рядки.

Для *цілих констант* тип можна не зазначати, оскільки за промовчанням константам призначається тип `int`, наприклад:

```
const B = 286; // інтерпретується як const int B=286
```

Окрім десяткових цілих констант, в C++ можна використовувати вісімкові (перед ними обов'язково ставиться 0) і шістнадцяткові (перед ними слід ставити 0x) цілі константи, наприклад:

```
const C=01, Q=0555, W=0777; // вісімкові константи
const k=0xFF, K=0x1A, v=0x23 // шістнадцяткові константи
```

Значення вісімкових констант утворюються з десяткових цифр без вісімок та дев'яток, позаяк ці цифри не входять до вісімкової системи числення. А шістнадцяткові константи утворюються з набору цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Приклади значень цілих констант:

Десяткові константи	Вісімкові константи	Шістнадцяткові константи
16	020	0x10
127	0117	0x2B
240	0360	0xF0

За потреби оголошення від'ємних цілих констант використовують знак «-» перед записом константи, який ще називають унарним мінусом, наприклад: -0x2A, -067, -16.

Для *дійсних констант* слід зазначити тип (`float`, `double` чи `long double`) і записати їхні значення з десятковою крапкою. Окрім звичної форми, дійсні константи можна записувати і в експоненціальній формі з рухомою крапкою, де десяткове дійсне число N записують у вигляді мантиси m і порядку p : $N = m \cdot 10^p$. Наприклад, число 373 можна подати як $0.373 \cdot 10^3$, при цьому мантиса $m = 3$, а порядок $p = 3$. Число 0.0002 можна записати як $0.2 \cdot 10^{-3}$ чи $2 \cdot 10^{-4}$. Запис у C++ цих чисел матиме вигляд 0.373e3 та 2e-4, де замість основи 10 використовується літера «e», після якої ставиться показник степеня. Число у формі

з рухомою крапкою може бути подане безліччю способів: $2 = 20 \cdot 10^{-1} = 0.2 \cdot 10^1 = 0.002 \cdot 10^3$ і т. ін. Для однозначного подання на мантису накладається обмеження: $0 \leq |m| < 1$, при дотриманні якого число є *нормалізованим*.

Приклади оголошення дійсних констант:

```
const float fi = -1.84;  
const double eps = 1e-4; // еквівалентно до eps = 0.0001
```

Символьна константа – один символ в одинарних лапках ‘*’*, наприклад:

```
const char d='+', a='ц', c='9';
```

Рядкова константа (літерал) – послідовність символів, включаючи великі та малі літери кирилиці й латиниці, а також цифри і пробіли, розташовані поміж подвійних лапок «*»*, наприклад:

```
const String^ S="Одеса – місто-герой!"; // рядок String  
const char* SS="ОНАЗ ім. О. С. Попова"; // рядок mny char*
```

Окрім літер і цифр, у символьних та рядкових константах можна задавати і керувальні символи: ‘*\n*’ (Enter), ‘*\0*’ (нуль-символ), ‘*\v*’ – вертикальна табуляція тощо. Значенням символьної константи є числовий код символу.

Зауважимо, що керувальні символи можуть входити до складу рядків, наприклад, при виведенні літерала "ОНАЗ *\n* ім. О. С. Попова" його частину "ОНАЗ" буде виведено в один рядок, а друга частина "ім. О. С. Попова" – в наступний рядок.

Якщо ключове слово **const** стоїть перед оголошенням структурованих типів (масив, структура, перерахування), це призводить до того, що кожен елемент також є константою, тобто значення йому може бути присвоєне лише один раз.

3.5 Правила записування арифметичних виразів

3.5.1 Операнди і вирази

Комбінація знаків операцій та операндів, результатом якої є конкретне значення, називається *виразом*. Знаки операцій визначають дії, які має бути виконано над операндами. Значення виразу залежить від розташування знаків операцій і круглих дужок у виразі, а також від пріоритету виконання операцій.

Операнд – це константа, літерал, ідентифікатор, виклик функції, індексний вираз, вираз вибору елемента чи більш складний вираз, сформований комбінацією операндів, знаків операцій і круглих дужок. Кожен операнд має тип.

Арифметичний вираз складається із операндів, арифметичних операцій (+ додавання, – віднімання, * множення, / ділення, % остача від ділення цілих чисел) і оператора присвоювання (=). Вираз, який завершується крапкою з комою, є **оператором**. Оператор задає закінчений опис певної дії. Виконання оператора виразу полягає в обчисленні виразу. Окремим випадком виразу є *порожній оператор* (;), який використовується, коли за синтаксисом оператор потрібен, а за змістом – ні (див. п. 4.3.2).

3.5.2 Арифметичні операції

Для записування виразів використовують відповідні операції. В C++ існують унарні, бінарні та тернарні операції, залежно від кількості операндів. Більшість операцій є бінарними, тобто мають два операнди, один з яких розміщується перед знаком операції, а другий – після. Наприклад, операція додавання «+» має два операнди: x та y й обчислює їхню суму. Унарні операції є ті, які мають лише один операнд. Наприклад, вираз $x++$ означає застосування до операнда x операції унарного інкременту (збільшення на 1). Тернарну умовну операцію ($?:$), яка має три операнди, буде розглянуто докладно у п. 4.3.7.

Вирази можуть бути арифметичними, символьними та логічними.

Арифметичні вирази – це сукупність числових констант, змінних і функцій, пов'язаних знаками арифметичних операцій і круглих дужок. Значенням арифметичних виразів є число. C++ підтримує арифметичні операції, наведені в табл. 3.3.

Таблиця 3.3

Арифметичні операції

Позначення	Операція	Типи операндів і результату	Приклади
+	додавання	арифметичний, вказівник	$x + y$
–	віднімання та унарний мінус	арифметичний, вказівник	$x - y$
*	добуток	арифметичний	$x * y$
/	ділення	арифметичний	x / y
%	остача від ділення цілих чисел	цілий	$i \% 6$
++	збільшення на одиницю (інкремент)	арифметичний, вказівник	$i++$; $++i$
--	зменшення на одиницю (декремент)	арифметичний, вказівник	$i--$; $--i$

Результатом операції обчислення остачі від ділення (%) є залишок від ділення першого операнда на другий. Операндами цієї операції мають бути цілі числа. Знак результату залежить від конкретної реалізації, переважно знак результату збігається зі знаком діленого. Якщо другий операнд дорівнює нулю, видається відповідне повідомлення.

```
int n = 49, m = 10, i, j, k, l;
i = n % m;           // i = 9
j = n % (-m);        // j = 9
k = (-n) % m;        // k = -9
l = (-n) % (-m);     // l = -9
```

Операції ++ (інкремент) та -- (декремент) є унарними, тобто мають лише один операнд. Операція ++ додає одиницю до операнда, операція -- віднімає одиницю від операнда. Ці операції можуть бути записані як праворуч, так і ліворуч операнда. Залежно від місця розміщення операції щодо операнда, розрізняють дві форми цих операцій: префіксну та постфіксну. У префіксній формі, в якій операцію розміщують перед операндом, наприклад: $++i$, $--j$, спочатку

збільшується чи зменшується на одиницю значення змінної, а вже потім ця змінна з її новим значенням бере участь в арифметичному виразі. У постфікській формі цих операцій, навпаки, операцію розміщують після операнда, наприклад: `i++`, `j--`, і у виразі спочатку використовується поточне значення цієї змінної, а потім збільшується чи зменшується її значення.

Три нижченаведені оператори дають однакові результати, але мають різницю при використанні у виразах:

```
i = i + 1; i += 1; ++i; i++;
```

Наприклад, в результаті виконання операторів

```
int j, i = 1;
j = i++ * i++;
```

значення змінної `i` дорівнюватиме 3, а змінної `j` – 1.

Якщо змінювати ці оператори в такий спосіб:

```
int j, i = 1;
j = ++i * ++i;
```

то результат буде іншим: значення `i` дорівнюватиме 3, а `j` дорівнюватиме 9.

Наведемо ще два приклади використання операції інкременту:

```
int k = 1, m, n;
n = k++; //порядок обчислень: 1) n=1, 2) k=k+1=2. Результат n=1, k=2
m = ++k; //порядок: 1) k=k+1=2+1=3, 2) n=k=3. Результат k=3, m=3;
int t = 1, s = 2, z, f;
z = (t++)*5; //порядок обчислень: 1)t*5, 2)t=t+1=2. Результат z=5, t=2
f = (++s)/3; //порядок: 1) s=s+1=3, 2) f=3/3=1. Результат s=3, f=1
```

Обчислення в арифметичних виразах виконуються зліва направо згідно з таким пріоритетом операцій:

- 1) стандартні функції, `++`, `--`;
- 2) множення (`*`), ділення (`/`), остача від ділення (`%`);
- 3) додавання (`+`) й віднімання (`-`).

Вирази у круглих дужках виконуються першочергово.

Пріоритет операцій наведено у додатку Б.

Для здобуття правильного результату слід дотримуватися таких правил записування арифметичних виразів в операторах C++:

- ✓ кожна команда (інструкція) має завершуватись крапкою з комою (`;`);
- ✓ мова C++ є чутлива до регістру, тобто `x` та `X` – це дві різні змінні;
- ✓ аргумент функції завжди записують у круглих дужках;
- ✓ знаки множення не можна пропускати (`3ab` → `3*a*b`);
- ✓ якщо знаменник чи чисельник має операції (`+`, `-`, `*`, `/`), то його слід записувати у круглих дужках;
- ✓ для записування раціональних дробів, у чисельнику чи знаменнику яких є числові константи, хоча б одну з цих констант слід записати як дійсне число із зазначенням десяткової крапки, наприклад, $\frac{2}{k}$ записують як `2.0/k` чи `2./k`;
- ✓ радикали (тобто корінь кубічний і вище) замінюють на дробові степені, наприклад, $\sqrt[3]{x+1}$ записують як `pow(x+1, 1./3)`;

✓ слід враховувати правила зведення типів, позаяк в арифметичних виразах можуть брати участь різнотипні дані й відбувається зведення типів.

3.5.3 Оператори присвоювання

Оператор присвоювання – основний оператор програмування – має формат:

<ім'я_змінної> = <вираз>;

Цей оператор обчислює значення *виразу* і надає здобуте значення *змінній*; при цьому слід враховувати відповідність типів виразу і змінної.

У мові C++, окрім простого присвоювання, є і складені операції присвоювання: += (присвоювання з додаванням), -= (присвоювання з відніманням), *= (присвоювання з множенням), /= (присвоювання з діленням) тощо (табл. 3.4). Наприклад, вираз $x += y$ є еквівалентний до виразу $x = x + y$, але записується компактніше і виконується швидше.

Таблиця 3.4

Операції присвоювання

Вигляд	Операція	Типи операндів і результату	Приклади
=	присвоювання	будь-які	$x = y$
+=	присвоювання з додаванням	арифметичні, вказівники, структури, об'єднання	$x += y$
-=	присвоювання з відніманням	арифметичні, вказівники, структури, об'єднання	$x -= y$
*=	присвоювання з множенням	арифметичні	$x *= y$
/=	присвоювання з діленням	арифметичні	$x /= y$
%=	присвоювання остачі від ділення цілих чисел	цілі	$x \% = y$
<=	присвоювання зі зсувом ліворуч	цілі	$x <= y$
>=	присвоювання зі зсувом праворуч	цілі	$x >= y$
&=	присвоювання з порозрядною операцією І	цілі	$x \& = y$
^=	присвоювання з порозрядною операцією виключне АБО	цілі	$x \wedge = y$
=	присвоювання з порозрядною операцією АБО	цілі	$x = y$

Особливістю звичайного оператора присвоювання є те, що він може використовуватись у виразах, наприклад:

if ((f = x - y) > 0) . . . ;

і допускає багатократне застосування, наприклад:

a = b = c = x * y;

Виконується ця команда справа наліво, тобто спочатку обчислюється значення виразу $x * y$, після чого це значення присвоюється c, потім b і лише потім a.

3.5.4 Зведення типів

Перетворення типів виконуються, якщо операнди, які входять до виразу, мають різні типи. Зведення типів здійснюється автоматично за правилом: менш точний тип зводиться до більш точного. Наприклад, якщо в арифметичному виразі беруть участь коротке ціле (`short`) і ціле (`int`), результат зводиться до `int`, якщо ціле і дійсне – до дійсного.

Розглянемо загальні правила зведення типів:

- 1) операнди типу `float` зводяться до типу `double`;
- 2) якщо один операнд `long double`, то другий теж зводиться до цього типу;
- 3) якщо один операнд `double`, а другий – `float` чи ціле число, то другий операнд перетворюється до типу `double`;
- 4) операнди цілих типів `char` та `short` зводяться до типу `int`;
- 5) цілі операнди типу `unsigned char` та `unsigned short` зводяться до типу `unsigned int`;
- 6) якщо один операнд типу `unsigned long`, то другий цілий операнд перетворюється до типу `unsigned long`;
- 7) якщо один операнд типу `long`, то другий зводиться до типу `long`;
- 8) якщо один операнд типу `unsigned int`, то другий цілий операнд зводиться до цього ж типу.

Отже, можна зауважити, що при обчисленні виразів операнди зводяться до типу того операнда, котрий має більший розмір, наприклад:

```
unsigned char  ch;  
double  ft, sd;  
unsigned long  n;  
int  i;  
sd = ft * (i + ch / n);
```

При виконанні цього присвоювання правила зведення використовуватимуться у такий спосіб. Операнд `ch` зводиться до `unsigned int` (правило 5). Після чого він зводиться до типу `unsigned long` (правило 6). За цим самим правилом `i` зводиться до `unsigned long`, і результат операцій у круглих дужках матиме тип `unsigned long`. Тоді він зводиться до типу `double` (правило 3), і результат всього виразу матиме тип `double`.

Розглянемо кілька правил зведення типів.

1) Результат операції ділення буде цілим числом, якщо ділене і дільник є цілими, і дійсним числом, якщо один з операндів є дійсного типу. Наприклад, результатом $2/3$ буде 0, а результатом $2.0/3$ чи $2./3$ – $0.666(6)$. Наведемо ще один приклад, коли обидва операнди є цілими змінними:

```
int  m = 2, n = 3;  
float a = m / n;    // У результаті a = 0
```

2) При присвоюванні остаточний результат зводиться до типу змінної, яка стоїть ліворуч « $=$ »; при цьому тип може як підвищуватися, так і знижуватись. Якщо тип лівого операнда є менш точним, аніж тип результату виразу, розміщеного праворуч оператора присвоювання, то можлива втрата точності чи то взагалі неправильний результат, наприклад:

```
float a = 2.8, b = 1.7;
int c = a * b;
```

Внаслідок виконання цих команд змінна `c` набуде значення 4, а не 4.76, тобто дійсну частину числа буде втрачено, позаяк `c` є цілою змінною.

3) Ще один приклад дає неправильний результат:

```
double a = 300, b = 200;
short c = a * b;
```

Після виконання цих команд змінна `c` набуде значення -5536 замість очікуваного 60 000. Це пов'язано з тим, що змінна типу `short` може зберігати значення не більше за 32767.

Зведення типів може бути неявним, при виконанні операцій та виклику функцій, чи явним, при виконанні операцій зведення типів.

Операцію явного зведення типів слід застосовувати, щоб уникнути помилок, подібних до вищенаведених прикладів. Ця операція має вигляд:

$\langle \text{тип} \rangle \langle \text{арифметичний\_вираз} \rangle$

тобто перед ім'ям змінної у дужках зазначається тип, до якого її слід перетворити. Наприклад, вищерозглянутий приклад до правила 1 буде обчислювати правильний результат, якщо записати його у такий спосіб:

```
int m = 2, n = 3;
double a = (double) m/n;    // У результаті a=0.66(6)
int i = 2; long l = 2; double d; float f;
d = (double) i * (double) l;
f = (float)d;
```

У цих прикладах величини `m`, `n`, `i`, `l`, `d` зводяться явно до зазначених у круглих дужках типів.

Потреба у зведенні типів виникає, наприклад, у разі, коли функція повертає вказівник на тип `void`, який слід присвоїти змінній конкретного типу для подальших операцій з нею:

```
float * w= (float *) malloc(25 * sizeof(float));
```

Явне зведення типу є джерелом можливих помилок, оскільки вся відповідальність за його результат покладається на програміста. Операцію явного зведення типів слід використовувати обережно, лише якщо іншого способу здобуття коректного результату немає. Наприклад, у виразі $z/(x+25)$, де `z`, `x` – цілі змінні, дробова частина втрачається. Запобігти цьому можна, якщо перетворити чисельник чи знаменник до дійсного типу. Для цього слід використати один з трьох способів: 1) приписати десяткову точку до числа 25. Вираз набуде вигляду $z/(x+25.)$; 2) домножити чисельник чи знаменник ліворуч на 1.0. Вираз набуде вигляду $1.0*z/(x+25)$; 3) явно зазначити тип результату $(double)z/(x+25)$.

Явне зведення типів дозволяє подати дані у зручному для програміста вигляді. Застосування явного зведення типів у наведеному нижче прикладі дозволяє здобути зображення ASCII-символів за їхнім порядковим номером:

```
for (int i=32; i <= 125; i++) cout << (char)i;
```

3.5.5 Математичні функції

Математичні функції широко використовуються для записування різних математичних залежностей та виразів. Список математичних функцій C++ наведено у табл. 3.5.

Таблиця 3.5

Основні математичні функції

Функція	Опис	Заголовний файл
<code>int abs(int i)</code>	модуль (абсолютне значення) цілого числа $x - x $	<code>stdlib.h</code>
<code>double fabs(double x)</code>	модуль дійсного числа $x - x $	<code>math.h</code>
<code>double sqrt(double x)</code>	корінь квадратний $-\sqrt{x}$	<code>math.h</code>
<code>double pow(double x, double y)</code>	піднесення x до степеня $y - x^y$	<code>math.h</code>
<code>double exp(double x)</code>	експонента e^x	<code>math.h</code>
<code>double log(double x)</code>	натуральний логарифм $-\ln(x)$	<code>math.h</code>
<code>double log10(double x)</code>	десятковий логарифм $-\lg(x)$	<code>math.h</code>
<code>double cos(double x)</code>	косинус $-\cos(x)$	<code>math.h</code>
<code>double sin(double x)</code>	синус $-\sin(x)$	<code>math.h</code>
<code>double tan(double x)</code>	тангенс $-\operatorname{tg}(x)$	<code>math.h</code>
<code>double acos(double x)</code>	арккосинус $-\arccos(x)$	<code>math.h</code>
<code>double asin(double x)</code>	арксинус $-\arcsin(x)$	<code>math.h</code>
<code>double atan(double x)</code>	арктангенс $-\operatorname{arctg}(x)$	<code>math.h</code>
<code>double atan2(double y, double x)</code>	арктангенс $-\operatorname{arctg}(y/x)$	<code>math.h</code>
<code>double cosh(double x)</code>	гіперболічний косинус $-(e^x + e^{-x})/2$	<code>math.h</code>
<code>double sinh(double x)</code>	гіперболічний синус $-(e^x - e^{-x})/2$	<code>math.h</code>
<code>double ceil(double x)</code>	округлення доверху: найменше ціле, не менше за x	<code>math.h</code>
<code>double modf(double x, double &z)</code>	виокремлює у дійсному числі x дробову частину і повертає у якості результату, а цілу частину числа доправляє на адресу вказівника $*z$	<code>math.h</code>
<code>double floor(double x)</code>	округлення донизу: найбільше ціле, не більше за x	<code>math.h</code>
<code>M_PI</code>	константа $\pi = 3.14159$	<code>math.h</code>
<code>M_E</code>	константа 2.71828182845904523536	<code>math.h</code>

До речі, константи **math** (наприклад, `M_PI` ($\pi \approx 3.14159$), `M_E` (число Ейлера $e \approx 2.71828$, яке є основою натуральних логарифмів) та багато інших) не вказані у стандартному C/C++. Щоб використовувати їх, слід спочатку визначити `_USE_MATH_DEFINES`, а тоді долучити **cmath** або **math.h**.

```
#define _USE_MATH_DEFINES
```

```
#include <math.h> // для C або #include <cmath> для C++
```

У платформі .NET існують свої відповідні константи `Math::PI` та `Math::E`. Крім того, в ній є зручна функція `Math::Log(x,N)`, яка дозволяє обчислити значення логарифма за будь-якою основою `N` (для обчислення такого логарифма в «чистому» C доведеться використовувати вираз $\log(x)/\log(N)$).

У табл. 3.6 наведено приклади записування арифметичних виразів.

Таблиця 3.6

Приклади записування виразів мовою C++

Математичний запис виразу	Запис виразу мовою C++
$y = \cos x^2 + \sin^2 x$	<code>y = cos(x *x)+ pow(sin(x), 2);</code>
$y = \sin^4 x$	<code>y = pow(sin(x), 4);</code>
$y = \sqrt[5]{x} \quad (\sqrt[5]{x} = x^{1/5})$	<code>y = pow(x, 1./5);</code>
$y = \frac{\sin(x - \pi) + 1}{e^x - 2.5x}$	<code>y = (sin(x - M_PI) + 1)/(exp(x)-2.5*x);</code>

3.6 Функції C++ генерування випадкових чисел

У табл. 3.7 наведено синтаксис і опис функцій C++ для генерування випадкових чисел.

Таблиця 3.7

Функції роботи з випадковими числами в C++

Прототип	Опис	Заголовний файл
<code>int random (int num);</code>	Повертає випадкове ціле додатне число в діапазоні від 0 до (num-1). Наприклад, команда <code>int x= random(100);</code> надасть змінній випадкове число в межах від 0 до 99	<code>stdlib.h</code>
<code>int rand(void);</code>	Повертає випадкове ціле число у діапазоні від 0 до символічної константи <code>RAND_MAX</code> . Ця константа визначена у файлі <code><stdlib.h></code> значенням 32767. Початкове значення генерованого числа задається звертанням до функції <code>srand()</code> . Для того щоб генерувати різні послідовності випадкових чисел, початкова точка має вибиратися випадково. Для цього можна використовувати функцію <code>randomize()</code> . Наприклад, команда <code>int x=rand() % 100;</code> надасть змінній випадкове число в межах від 0 до 99	<code>stdlib.h</code>
<code>void randomize(void);</code>	Ініціалізує генератор випадкових чисел, використовуючи поточний час комп'ютера	<code>stdlib.h,</code> <code>time.h</code>

Закінчення табл. 3.7

Прототип	Опис	Заголовний файл
<code>void srand (unsigned seed);</code>	Встановлює початкове число <code>seed</code> для генерованої за допомогою функції <code>rand()</code> послідовності випадкових чисел. Якщо <code>seed=1</code>, генератор випадкових чисел ініціалізується значенням за промовчанням. Наприклад, команди <pre>int i; time_t t; srand((unsigned) time(&t)); for(i=0; i<10; i++) printf("%d\n", rand() % 100);</pre> генеруватимуть послідовність з 10-ти випадкових чисел. Для використання цієї функції слід долучити бібліотеки <code>time.h</code> та <code>stdlib.h</code>	<code>stdlib.h</code>

Приклади програм із застосуванням наведених функцій дивіться у подальших підрозділах, а саме приклади: 4.24, 4.48, 4.51, 4.64, 6.10, 12.6.

Питання та завдання для самоконтролю

- 1) Охарактеризувати мову C++ і середовище Microsoft Visual C++.
- 2) Що називають лексемами у програмуванні?
- 3) Як записуються коментарі в C++?
- 4) Що відбувається при оголошенні змінної?
- 5) З яких символів може складатися ім'я (ідентифікатор) змінної?
- 6) Що визначає тип даних?
- 7) Які базові типи даних C++ Вам відомі?
- 8) Що означають специфікатори `signed` та `unsigned` і до яких типів даних їх можна застосовувати?
- 9) Яких значень можуть набувати змінна логічного типу `bool`?
- 10) Коли використовується тип даних `void`?
- 11) Назвати призначення операції `sizeof`.
- 12) В який спосіб можна оголошувати власні типи?
- 13) Які особливості оголошування констант у C++?
- 14) Який тип надається константам за промовчанням?
- 15) Що називають операндом, виразом, оператором?
- 16) Перелічити арифметичні операції C++.
- 17) Яких значень набудуть змінні `j` та `i` після обчислення для `int j, i=2`:
 - а) `j = i++ + i++;`
 - б) `j = ++i + ++i;`
 - в) `j = i++ + ++i;`
 - г) `j = ++i + i--;`
- 18) Записати числові константи `0.2731e3`, `2.5e-4` у фіксованому форматі.
- 19) Записати числа `0.0001`, `2500` у формі з рухомою крапкою.

- 20) Обчислити вирази для `int a=5, b=7, c=13, d=3`:
- а) $a * b / (c - d)$;
 - б) $(b + c) \% (a + d)$
- 21) Якого значення набуде змінна `y` після обчислення:
- а) $y = (1 + 2) / (3 + 2)$;
 - б) $y = 1 + 2 / 3 + 2$;
 - в) $y = 1 + 2. / 3 + 2$;
 - г) $y = (1 + 2) / (3. + 2)$;
- 22) Записати вирази $y = x^3 + \sin 2x$, $y = \frac{1 + \sin x^2}{\sqrt{5a + \ln x}}$ засобами C++.
- 23) У чому полягає відмінність арифметичних операції / та %?
- 24) Чому дорівнюватиме результат обчислення виразів: $22/5$ та $22\%5$?
- 25) Чому результат обчислення виразу $1/4$ дорівнюватиме 0? Що треба зробити, щоб результат був 0.25?
- 26) Якими правилами визначається послідовність виконання операцій у виразах?
- 27) Які дії виконує оператор присвоювання?
- 28) Коли використовують операцію явного зведення типів?
- 29) В який спосіб можна долучити бібліотеку математичних функцій?
- 30) Записати команду для надання змінній `x` випадкового значення у межах від -50 до 50.

РОЗДІЛ 4

ПРОГРАМУВАННЯ БАЗОВИХ АЛГОРИТМІВ

Види та специфіку базових алгоритмів організації обчислень було розглянуто в підрозд. 1.3.

4.1 Програмування лінійних алгоритмів

Лінійним називається алгоритм, в якому всі дії, від першої до останньої, виконуються послідовно. Іноді таку структуру називають просто *послідовністю*. Зазвичай у програмній реалізації послідовності використовують оператори введення вхідних даних, оператори обчислення та оператори виведення результатів.

Приклади програм мовою C++ з лінійною структурою

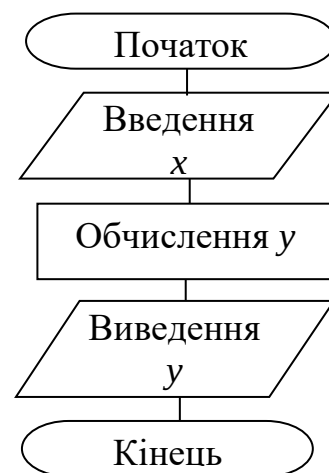
Приклад 4.1. Розробити схему алгоритму і створити програмний проект для обчислення $y = \frac{0.2x^2 - x}{(\sqrt{3} + x)(1 + 2x)} + \frac{2(x-1)^3}{\sin^2 x + 1}$, де x – число, яке слід ввести.

Розв'язок. Послідовність виконання завдання:

- 1) Запустити Visual Studio.
- 2) Виконати команду *Создать проект / Visual C++ / CLR / Консольное приложение CLR*. Не закриваючи вікно *Создать проект*, натиснути кнопку *Обзор*, вибрати папку для зберігання та задати ім'я програмного проекту. Після цього натиснути кнопку *ОК* та приступити до написання програмного коду.
- 3) Перед запуском на виконання програми слід її зберегти командою *Файл / Сохранить все* (чи то через ярлик , чи клавішами [Ctrl]+[Shift]+[S]).
- 4) Для запуску програми слід виконати команду меню *Отладка / Начать отладку* чи то натиснути на клавіатурі кнопку [F5].

Текст програмного коду та схема алгоритму:

```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{ setlocale(0, ".1251");
  double y, x;
  cout<<"Введіть значення x= "; cin>>x;
  y=(0.2*x*x-x)/((sqrt(3.)+x)*(1+2*x)) +
    2*pow(x-1,3)/(pow(sin(x),2)+1);
  cout<<"Результат y= "<< y <<endl;
  system ("pause>>void");
  return 0;
}
```



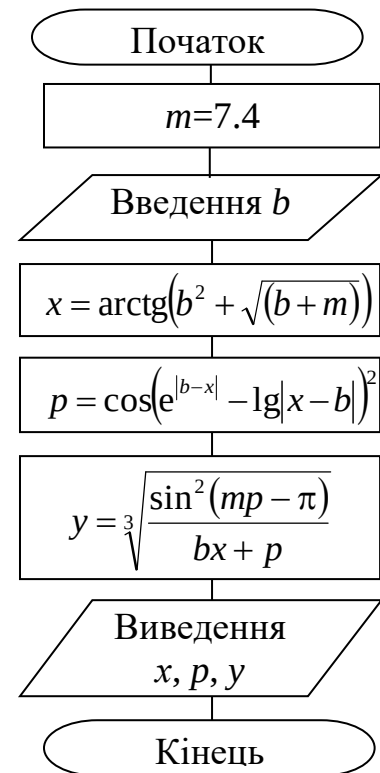
Результати роботи консольної програми:

```
Введіть значення x= 2.15
Результат y= 1.72928
```

Приклад 4.2. Розробити схему алгоритму і запрограмувати обчислення $y = \sqrt[3]{\frac{\sin^2(mp - \pi)}{bx + p}}$; $x = \arctg(b^2 + \sqrt{(b + m)})$ і $p = \cos(e^{|b-x|} - \lg|x-b|)^2$. Значення $m = 7.4$ задати як константу, а довільне значення змінної b слід ввести.

Текст програми та схема алгоритму:

```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{
    setlocale(0, ".1251");
    const double m=7.4, pi=3.1415;
    double b, x, p, y;
    cout<<"Введіть значення b= "; cin>>b;
    x=atan(b*b+sqrt(b+m));
    p=cos(pow(exp(fabs(b-x))-
        log10(fabs(x-b)),2));
    y=pow(pow(sin(m*p-pi),2)/(b*x+p),1./3);
    cout<<"Результати:\n x= "<< x <<endl;
    cout<<" p= "<< p <<endl;
    cout<<" y= "<< y <<endl;
    system ("pause>>void");
    return 0;
}
```



Результати роботи консольної програми:

```
Введіть значення b= 8.4
Результати:
x= 1.55738
p= -0.128333
y= 0.3709956
```

Приклад 4.3. Обчислити ємність конденсатора за формулою $C = \frac{\varepsilon_0 \cdot \varepsilon \cdot S}{d}$,

де $\varepsilon_0 = 8.85 \cdot 10^{-12}$ Ф/м, $\varepsilon = 2.8$ Ф/м, S – площа кожної пластини конденсатора, d – відстань між пластинами, значення яких ввести з клавіатури.

Текст програмного коду:

```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{ setlocale(0, ".1251");
```

```
double d, S, C;
const double E0 = 8.85e-12, E = 2.8;
cout<< "Введіть значення відстані між пластинами d= ";
cin>>d;
cout<< "Введіть значення площі пластини конденсатора S= ";
cin>>S;
C=(E0*E*S)/d;
cout<<"Ємність конденсатора C= "<< C <<endl;
system ("pause>>void");
}
```

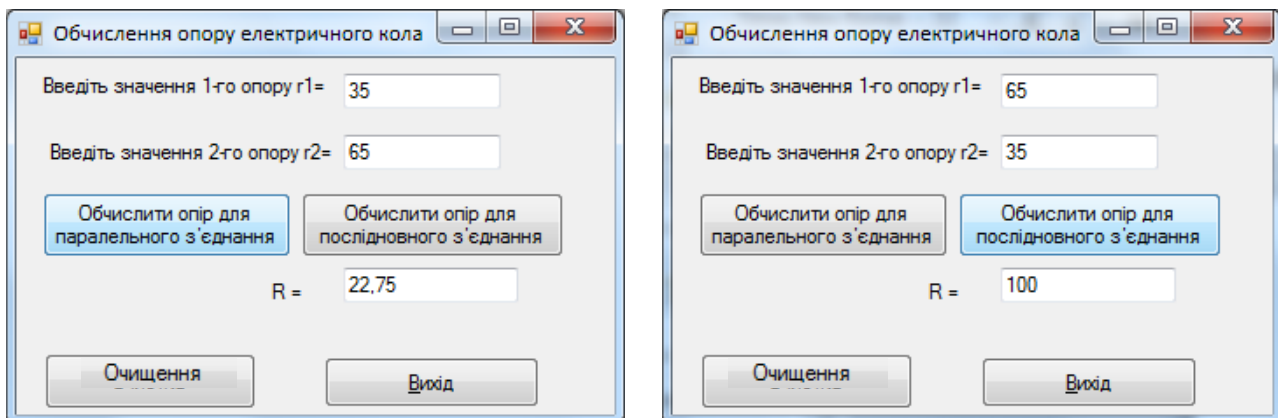
Результати роботи консольної програми:

```
Введіть значення відстані між пластинами d= 0.0004
Введіть значення площі пластини конденсатора S= 50000
Ємність конденсатора C= 0.0030975
```

Приклад 4.4. Створити проект для обчислення опору електричного кола, яке складається з двох поєднаних опорів двома варіантами: 1) за паралельного з'єднання опорів та 2) за послідовного з'єднання опорів в електричному колі.

Розв'язок. Нагадаємо, що формула для обчислення опору електричного кола за паралельного з'єднання опорів має вигляд $R = \frac{r1 \cdot r2}{r1 + r2}$, а за послідовного з'єднання опорів – $R = r1 + r2$.

Обчислення для двох випадків сформуємо в програмі для двох окремих командних кнопок на формі:



Для створення програмного проекту з формою слід запустити Visual Studio та виконати команду *Создать проект / Visual C++ / CLR / Приложение Windows Forms*. Не закриваючи вікно *Создать проект*, натиснути кнопку *Обзор*, вибрати папку для зберігання та задати ім'я програмного проекту. Натиснути кнопку *OK* та приступити до створення форми і написання програмного коду.

Текст програми:

```
// Обчислити для паралельного з'єднання
private: System::Void button1_Click(System::Object^sender, System::EventArgs^e)
{ double r1, r2, R;
  r1 = Convert::ToDouble(textBox1->Text); // значення 1-го опору r1
  r2 = Convert::ToDouble(textBox2->Text); // значення 2-го опору r2
```

```

    R = r1*r2/(r1+r2);
    textBox3->Text = Convert::ToString(R);
}
// Обчислити для послідовного з'єднання
private: System::Void button2_Click(System::Object^sender, System::EventArgs^e)
{
    double r1, r2, R;
    r1=Convert::ToDouble(textBox1->Text); // значення 1-го опору r1
    r2=Convert::ToDouble(textBox2->Text); // значення 2-го опору r2
    R=r1+r2;
    textBox3->Text=Convert::ToString(R);
}
// Очищення
private: System::Void button3_Click(System::Object^sender, System::EventArgs^e)
{
    textBox1->Clear(); textBox2->Clear(); textBox3->Clear();
}
// Вихід
private: System::Void button4_Click(System::Object^sender, System::EventArgs^e)
{
    Close();
}

```

Приклад 4.5. Розробити схему алгоритму і створити програму в онлайн-

компіляторі для обчислення $y = \frac{\left(\sin^3 x^{1.5a} + \arctg^3 \sqrt{|x + \sqrt{ax}|} \right)^4}{\left(\ln(|x| + e^{\sqrt{\lg^3(0.5x^2)^3}}) + x^2 \right)}$, де x, a – довільні

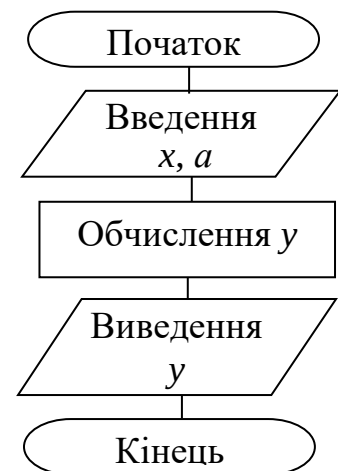
змінні, значення яких слід ввести.

Розв'язок. Текст програми, блок-схема розв'язування задачі і робочий вигляд вікна онлайн-компілятора мають вигляд:

```

#include <iostream>
#include <math.h>
using namespace std;
int main()
{
    double a, x, y;
    cout<< "Input 2 numbers:\nx = ";    cin>>x;
    cout<< "a = ";    cin>>a;
    y=pow(pow(sin(pow(x,1.5*a)),3) +
        pow(atan(sqrt(fabs(x+sqrt(a*x))))),3),4) /
        (log(fabs(x) + exp(sqrt(pow(log10(pow(0.5* x*x,
        3)), 3)))) + x*x);
    cout<<"y = "<<y;
}

```



Наприкінці цього програмного коду немає команд зупину, позаяк код виконується в онлайн-компіляторі, то консольне вікно з результатами не може закритися, а отже в цих командах немає потреби.

C++ shell

```

1 // Example program
2 #include <iostream>
3 #include <math.h>
4 using namespace std;
5 int main()
6 {
7     double a, x, y;
8     cout<< "Input 2 numbers:\nx= ";
9     cin>>x;
10    cout<< "a= ";
11    cin>>a;
12    y=pow(pow(sin(pow(x, 1.5*a)), 3) + pow(atan(sqrt(fabs(x + sqrt(a*x)))), 3), 4) / (log(fabs(x) + exp(sqrt(pow(log10(pow(0.5* x*x, 3))), 3))) + x*x);
13    cout<<"y = "<<y;
14 }

```

Get URL Run

options compilation execution

Input 2 numbers:
x= 10
a= 2.5
y = 0.246661

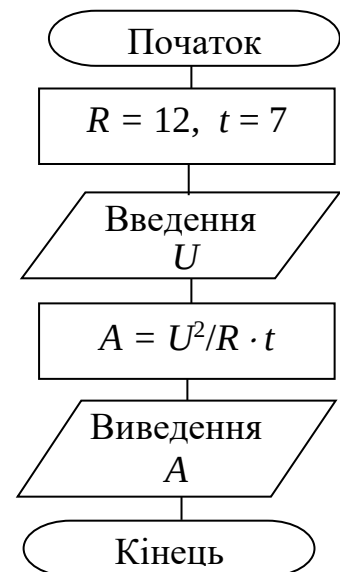
Приклад 4.6. Обчислити роботу електричного струму на ділянці кола за формулою $A = \frac{U^2}{R} \cdot t$, де опір $R = 12$ Ом, час $t = 7$ с, а значення напруги U ввести з клавіатури.

Текст програмного коду та блок-схема:

```

#include <iostream>
#include <math.h>
using namespace std;
int main()
{
    float R=12, A, t=7, U;
    cout<< "Input U = ";
    cin>>U;
    A=pow(U,2)/R *t;
    cout<<"A = "<<A;
    return 0;
}

```



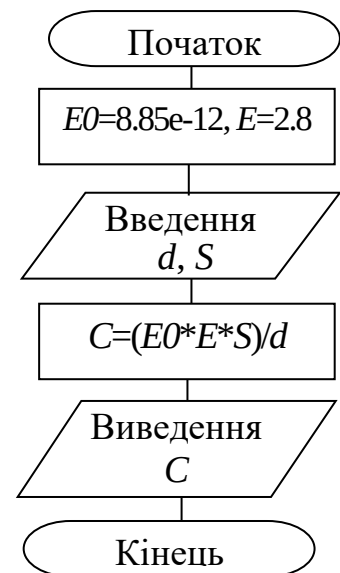
Результати роботи консольної програми:

Input U = 12
A = 84

Приклад 4.7. Обчислити ємність конденсатора за формулою $C = \frac{\varepsilon_0 \cdot \varepsilon \cdot S}{d}$, де $\varepsilon_0 = 8.85 \cdot 10^{-12}$ Ф/м, $\varepsilon = 2.8$ Ф/м, S – площа кожної пластини конденсатора, d – відстань між пластинами, значення яких ввести з клавіатури.

Текст програмного коду:

```
#include <iostream>
using namespace std;
int main()
{
    const double E0=8.85e-12, E=2.8;
    double d, S, C;
    cout<< "Input d = ";    cin>>d;
    cout<< "Input S = ";    cin>>S;
    C=(E0*E*S)/d;
    cout<<"C = "<<C;
    return 0;
}
```



Результати роботи консольної програми:

```
Input d = 0.00001
Input S = 5000
C = 0.01239
```

Приклад 4.8. Обчислити ослаблення T -образного резисторного атенюатора за формулою

$$A = 20 \cdot \lg \frac{(1 + a \cdot R1 / R2) \cdot b \cdot R3 + R1}{C},$$

для якої значення опорів $R1, R2, R3, Z1, Z2$ ввести з клавіатури, а частотні коефіцієнти обчислити за формулами:

$$a = 1 + \frac{Z1}{R1}; \quad b = 1 + \frac{Z2}{R2}; \quad C = Z1 + Z2.$$

Текст програмного коду:

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
    float R1,R2,R3,Z1,Z2,a,b,C,A;
    cout<< "Input R1 = ";    cin>>R1;
    cout<< "Input R2 = ";    cin>>R2;
    cout<< "Input R3 = ";    cin>>R3;
    cout<< "Input Z1 = ";    cin>>Z1;
    cout<< "Input Z2 = ";    cin>>Z2;
    a = 1+(Z1/R1);    // Обчислення частотних коефіцієнтів a, b, C
    b = 1+(Z2/R2);    C = Z1+Z2;
    A = 20*log10(((1+a*R1/R2)*b*R3+R1)/C); // Обчислення затухання
    cout<<"\nA = "<<A;    // Виведення результату
    return 0;
}
```

Результати роботи консольної програми:

```
Input R1 = 65
Input R2 = 75
Input R3 = 20
Input Z1 = 20
Input Z2 = 10
```

A = 11.5464

Приклад 4.9. Створити програму для обчислення сили струму електричного кола залежно від значень опору та напруги, які введе користувач.

Текст програмного коду:

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
    double U, R, I;
    cout<< "Input Voltage U= ";    cin>> U;
    cout<< "Input Resistance R= "; cin>> R;
    I = U/R;
    cout<<"Result Current I= " << A << " Amper"<< endl;
    getch();
    return 0;
}
```

Результати роботи консольної програми:

```
Voltage U= 24
Resistance R= 48
Result Current I= 0.5 Amper
```

4.2 Програмування розгалужених алгоритмів

4.2.1 Розгалужені алгоритми

Алгоритм називається *розгалуженим*, якщо певні його команди (інструкції) можуть не виконуватись жодного разу. Це відрізняє такі алгоритми від лінійних. Вочевидь, деякі команди не виконуються за певних вхідних умов, але можуть виконуватися за інших умов. Якщо за аналогією проаналізувати повсякденні життєві ситуації, то можна віднайти численну кількість умов, які розпочинаються словом «якщо». Наприклад, «*якщо на вулиці йде дощ, я залишуся вдома, якщо – ні, то поїду на пікнік*» або «*якщо на вулиці дощ, треба взяти парасоль*».

Отже, *розгалуженням* називається вибір програмою тієї чи іншої низки команд залежно від того, чи виконується певна умова. При цьому спрацьовує лише одна з гілок алгоритму.

Для програмної реалізації таких обчислень слід використовувати оператори передавання керування, котрі дозволяють змінювати порядок виконання операторів програми. У мові C++ для цього передбачено інструкції: безумовного переходу – **goto**, умовного переходу – **if** та вибору варіанта – **switch**. Для записування умови переходу слід використовувати логічні (булеві) вирази.

4.2.2 Оператор безумовного переходу goto

Оператор **goto** (перейти до) дозволяє передавати керування у будь-яку точку коду (програми), котру позначено спеціальною міткою.

Синтаксис оператора **goto**:

```
goto <мітка>;
```

Мітку записують перед оператором, на який слід передати керування, і відокремлюють від цього оператора символом двокрапки (:). Мітки в мові C++ не оголошують. У якості мітки може застосовуватись сполучення будь-яких латинських літер та цифр, але розпочинатися мітка повинна з літери, наприклад: **start**, **M1**, **a**, **second**.

Застосовувати цей оператор слід вельми обережно і помірковано, особливо при переході всередину блока чи циклу, позаяк це може призвести до непередбачуваних помилок. Тому в C++ оператор **goto** застосовується рідко і вважається застарілим. А застосування структурного й об'єктно-орієнтованого підходів до програмування дозволяє повністю відмовитись від застосування операторів **goto**. Однак на практиці часто трапляються випадки, коли цей оператор значно спрощує код програми (див. приклад програми 4.26 у п. 4.2.8).

4.2.3 Операції відношення та логічні операції

Операції відношень (<, >, <=, >=, == (дорівнює), != (не дорівнює)) порівнюють два вирази і видають значення 1 (**true** – істина – «так») або 0 (**false** – хибність – «ні»). Типом результату є **int** (чи **bool**).

Наприклад:

<code>x < y</code>	<code>/* вираз 1 */</code>	<code>x >= y</code>	<code>/* вираз 4 */</code>
<code>y > x</code>	<code>/* вираз 2 */</code>	<code>x == y</code>	<code>/* вираз 5 */</code>
<code>x <= y</code>	<code>/* вираз 3 */</code>	<code>x != y</code>	<code>/* вираз 6 */</code>

Якщо `x`, `y` є однакові, то вирази 3, 4 та 5 мають значення 1, а вирази 1, 2 і 6 – значення 0.

Умову для перевірки певної цілої змінної `k` на ненульове значення `if(k!=0)` можна записати простіше – як `if(k)`, оскільки ненульове значення розцінюється як **true**. Тоді умову `if(k == 0)` можна записати як `if(!k)`.

Варто зауважити, що у C++ результатом логічного виразу може бути і цілочисельне арифметичне значення. При цьому значення 0 розцінюється як **false**, а кожне ненульове значення – як **true**. Розглянемо приклади:

```
int tr = (105<=109);  
int fal = (105>109);
```

Унаслідок виконання цих операторів змінна `tr` набуде значення 1, а змінна `fal` – 0.

Логічні змінні можуть набувати значень: **true** (істина) чи **false** (хибність). Вельми часто ці значення позначають цифрами 1 та 0 (1 – істина, 0 – хибність). Логічний тип ще називають *булевим*, від прізвища англійського математика Джорджа Буля, засновника математичної логіки. При оголошенні змінних булевого типу їх позначають як **bool**, наприклад: **bool m**;

Логічна операція – дія, яка виконується над логічними змінними, її результат є 1 (**true**) або 0 (**false**). Логічні операції обчислюють кожен операнд з огляду на його еквівалентність нулю.

Базові логічні операції:

|| – логічне додавання (операція «АБО», *диз'юнкція*), результатом виконання якого є значення 1 (**true**), якщо хоча б один з операндів має ненульове значення. Якщо перший операнд має ненульове значення, то другий операнд вже не обчислюється;

&& – логічне множення (операція «І», *кон'юнкція*), результатом виконання якого є 1, якщо обидва операнди мають ненульові значення, у решті випадків результат – 0. Якщо значення першого операнда дорівнює 0, то другий операнд не обчислюється;

! – логічне заперечення (операція «НЕ», *інверсія*) виконується над однією логічною змінною, результатом чого є значення **true** (1), якщо початковим значенням було **false** (0), і **false** (0), – якщо початковим значенням було **true** (1), наприклад:

```
bool w, q = true;
w = !q;           // Змінна w набуде значення false
int t, z = 0;
t = !z;           // Змінна t набуде значення 1
```

З логічних змінних за допомогою логічних операцій та дужок (для зазначення порядку дій) будуються логічні вирази. Результати застосування цих операцій до булевих значень наведено в табл. 4.1. Операнди логічних операцій можуть бути цілого, дійсного типу чи типу вказівника, при цьому в кожній операції можуть брати участь операнди різних типів. Операнди логічних виразів обчислюються зліва направо. Якщо значення першого операнда вистачає, щоб визначити результат операції, то другий операнд не обчислюється. Логічні операції не спричиняють стандартних арифметичних перетворень. Вони оцінюють кожен операнд з огляду на його еквівалентність нулю. Результатом логічної операції є 0 чи 1.

Таблиця 4.1

Булеві операції

A	B	A B	A && B	! A
0	0	0	0	1
0	1	1	0	1
1	0	1	0	0
1	1	1	1	0

Обчислення в логічних виразах виконуються зліва направо у відповідності з таким *пріоритетом* операцій: спочатку виконується операція “НЕ”, потім “І”, а вже потім “АБО”.

За допомогою логічних змінних і операцій будь-яке висловлювання можна формалізувати, тобто замінити логічною формулою (логічним виразом). Наприклад, $F(A, B) = A \& B \vee A$ – логічна функція двох змінних A і B .

Значення логічної функції для різних поєднань значень вхідних змінних – або, як це інакше називають, наборів вхідних змінних – зазвичай задаються спеціальною таблицею. Така таблиця називається **таблицею істинності**.

Наведемо таблицю істинності основних логічних операцій (табл. 2)

Таблиця 2

Таблиця істинності основних логічних операцій

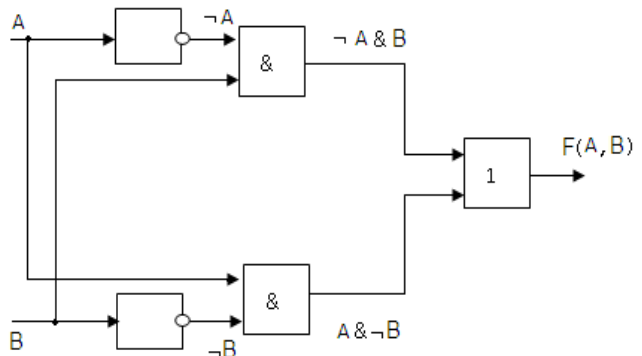
A	B	$\bar{A} (\neg A)$	$A \wedge B (A \& B)$	$A \vee B$	$A \oplus B$
1	1	0	1	1	0
1	0	0	0	1	1
0	1	1	0	1	1
0	0	1	0	0	0

На базі цієї таблиці можна складати таблиці істинності для складніших формул.

Приклад 4.10. Скласти таблицю істинності для виразу $C = \neg A \wedge B \vee A \wedge \neg B$, записати його мовою C++ та скласти відповідну програму.

Таблиця істинності для виразу

A	B	$\neg A \wedge B$	$A \wedge \neg B$	C
0	0	1 && 0 = 0	0 && 1 = 0	0 0 = 0
0	1	1 && 1 = 1	0 && 0 = 0	1 0 = 1
1	0	0 && 0 = 0	1 && 1 = 1	0 1 = 1
1	1	0 && 1 = 0	1 && 0 = 0	0 0 = 0



Вираз $C = \neg A \wedge B \vee A \wedge \neg B$ мовою C++ матиме вигляд:

$C = (!A \&\& B) \mid\mid (A \&\& !B);$

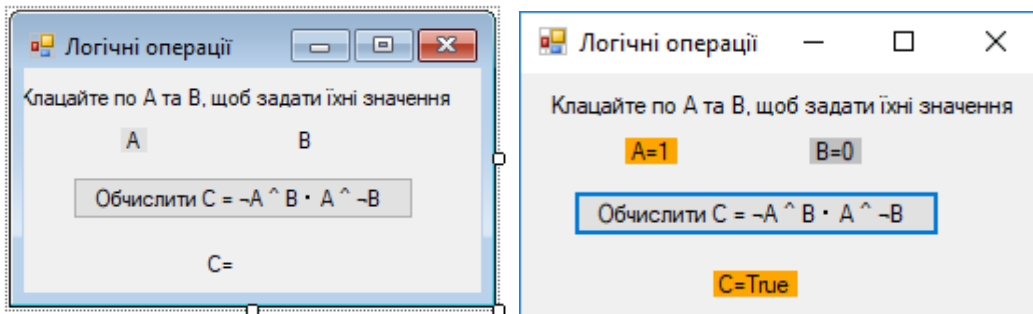
Текст програмного коду:

```
#include <iostream>
using namespace std;
int main()
{
    bool A, B, C;
    cout << "Input two boolean numbers (0 or 1): ";
    cin >> A >> B;
    C = (!A && B) || (A && !B);
    cout << "C = " << C;
    system("pause");
    return 0;
}
```

Результати роботи консольної програми:

Input two boolean numbers (0 or 1): 1 0
C = 1

Цей самий приклад можна реалізувати і з формою, наприклад так:



Ліворуч показано вигляд форми під час конструювання форми, а праворуч – під час роботи проекту.

Програмний код:

```
static bool x=true, y=1; // глобальні змінні
private: System::Void label1_Click(System::Object^ sender, System::EventArgs^ e)
{
    x = !x;
    if (x) { label1->BackColor = Color::Orange; label1->Text="A=1";}
    else { label1->BackColor = Color::Silver; label1->Text="A=0";}
}
private: System::Void label2_Click_1(System::Object^ sender, System::EventArgs^ e)
{
    y = !y;
    if (y) { label2->BackColor = Color::Orange; label2->Text="B=1";}
    else { label2->BackColor = Color::Silver; label2->Text="B=0";}
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    bool a,b,c;
    if (label1->Text=="A=1") a=1; else a=0;
    if (label2->Text=="B=1") b=1; else b=0;
    c = (!a && b) || (a && !b);
    if (c) label3->BackColor = Color::Orange;
    else label3->BackColor = Color::Silver;
    label3->Text="C="+c;
}
```

Логічні побітові операції виконуються над цілими числами і опрацьовують число побітово. До цих операцій належать: побітове логічне заперечення ($\sim$), побітове логічне множення «І» ($\&$), побітове логічне додавання «АБО» ($\|$), побітове логічне «виключне АБО» ($\wedge$) (табл. 4.2). Операнди логічних операцій мають бути будь-якого цілого типу.

Таблиця 4.2

Логічні побітові операції

Позначення	Операція	Приклади	
~	інверсія, НЕ	~X	~1011 = 0100
&	I	X & Y	1011 & 1010 = 1010
	АБО	X Y	1011 1010 = 1011
^	виключне АБО	X ^ Y	1011 ^ 1010 = 0001
<<	зсув ліворуч	X << 2	1011 << 2 = 1100
>>	зсув праворуч	Y >> 2	1011 >> 2 = 0010

Зауважимо, що в наведених у табл. 4.2 прикладах для простоти та наочності ми обмежились лише чотирма розрядами двійкових чисел.

Операція побітового логічного заперечення «НЕ» (~) інвертує кожен біт операнда: 0 – на 1, 1 – на 0, тобто продукує двійкове доповнення свого операнда ($\sim 1010_2 = 0101_2$), наприклад:

```
short x = 987;    unsigned short y = 0xAAAA;
y = ~y;           // У результаті y = 0x 5555 (шістнадцяткове)
if (!(x < y));     // Результат виразу 0 (false)
```

Операція побітового логічного множення «I» (&) порівнює кожен біт першого операнда з відповідним бітом другого операнда і, якщо обидва порівнюваних біти є одиницями, то відповідний біт результату встановлюється в 1, інакше – в 0. Наприклад, $1010_2 \& 1001_2 = 1000_2$.

Операція побітового логічного додавання «АБО» (|) порівнює кожен біт першого операнда з відповідним бітом другого операнда, і, якщо хоча б один (чи обидва) з них дорівнює 1, то відповідний біт результату встановлюється в 1, інакше – біт результату дорівнює 0. Наприклад, $1010_2 | 1001_2 = 1011_2$.

Операція побітове логічне «виключне АБО» (^) порівнює кожен біт першого операнда з відповідним бітом другого операнда і, якщо обидва опрацьовувані біти мають однакове значення, результат дорівнює 0, у решті випадків – 1. Наприклад, $1010_2 \wedge 1001_2 = 0011_2$.

Наведемо приклади виконання побітових логічних операцій.

```
short i=0x45FF, // i= 0100 0101 1111 1111
r, j=0x00FF;    // j= 0000 0000 1111 1111
r = i ^ j;       // r=0x4500 = 0100 0101 0000 0000
r = i | j;       // r=0x45FF = 0100 0101 0000 0000
r = i & j        // r=0x00FF = 0000 0000 1111 1111
short n, i = 0xAB00, j = 0xABCD;
n = i & j;       // У результаті n = AB00 (шістнадцяткове)
n = i | j;       // У результаті n = ABCD (шістнадцяткове)
n = i ^ j;       // У результаті n = CD (шістнадцяткове)
```

Операції побітового зсуву >> і << виконують зсув бітів лівого операнда на кількість розрядів, зазначену правим операндом, відповідно праворуч чи ліворуч. Значення бітів, яких бракує, доповнюються нулями. При виконанні операції зсуву можуть втрачатися старші чи молодші розряди:

$\gg$ – побітовий зсув праворуч на задану кількість бітів ($65 \gg 2 = 16$, оскільки $65 = 41_{16} = 0100\ 0001_2$ після зсуву праворуч становить $0001\ 0000_2 = 10_{16} = 16$). Операція є еквівалентна до цілочисельного поділу на 2 у відповідному степені ($65 / 4 = 16$);

$\ll$ – побітовий зсув ліворуч ($13 \ll 2 = 52$, позаяк $13 = D_{16} = 1101_2$ після зсуву ліворуч становить $11\ 0100_2 = 34_{16} = 52$). Операція є еквівалентна до цілочисельного множення на 2 у відповідному степені ($13 * 4 = 52$). Слід пам'ятати, що при цьому є можливе втрачання старших розрядів.

Наведемо кілька прикладів роботи операцій зсуву.

```
int  c = 12, m, n;
m = c >> 2;           // m=6
n = c << 2;           // n=24
c = n >> 3;           // c=24/23=24/8=3
int  i = 0x1234, j, k;
k = i << 4;           // k=0x0234
j = i << 8;           // j=0x3400
i = j >> 8;           // i=0x0034
unsigned int x = 0x00AA, y = 0x5500, z;
z = (x << 8) + (y >> 8);
```

В останньому прикладі x зсувається ліворуч на 8 позицій, а y – праворуч на 8 позицій. Результати зсувів додаються, утворюючи величину $0xAA55$, котра присвоюється z .

Застосовування операцій $\gg$ та $\ll$ по чергово до однієї й тієї самої змінної може спричинити змінення її значення через втрату розрядів.

Перетворювання, виконувані операціями зсуву, не забезпечують опрацювання ситуацій переповнення і втрати знаків. Знак першого операнда після виконання операції зберігається. Результат операції зсуву є невизначений, якщо другий операнд буде від'ємний.

Побітові операції є зручні для організації зберігання у стислому вигляді інформації про стан on/off (ввімкнено/вимкнено). В одному байті можна зберігати 8 таких «прапорців». Якщо змінна ch є сховищем таких «прапорців», то перевірити, чи є «прапорець» третього біту у стані on, можна у такий спосіб:

```
if (ch & 4) . . .
```

Ця перевірка базується на двійковому поданні числа $4 = 0000\ 0100$.

Зауважимо, що вираз $a \wedge a$ завжди повертає значення 0, а вираз $a \wedge b \wedge a$ – значення b . Ці закономірності часто застосовуються у растровій графіці.

Приклад 4.11. Розробити програмний код для виконання побітових логічних операцій та перевірити правильність виконаних обчислень:

- 1) інвертувати усі біти числа;
- 2) встановити (задати значення 1) для 4-го біта числа;
- 3) визначити (вибрати окремо) значення 3-го біта числа;
- 4) скинути (встановити в 0) 0-й біт числа;
- 5) інвертувати 2-й біт числа;

б) логічний побітовий зсув на 2 біти праворуч.

Для усіх здобутих двійкових чисел виконати переведення до шіснадцяткової та десяткової систем числення.

Розв'язок: В якості перевірки правильності виконаних у програмі обчислень візьмемо як приклад число 105 і переведемо його спочатку до двійкової системи числення:

$$\begin{array}{rcl}
 105 : 2 & = & 52 \text{ (1)} \\
 52 : 2 & = & 26 \text{ (0)} \\
 26 : 2 & = & 13 \text{ (0)} \\
 13 : 2 & = & 6 \text{ (1)} \\
 6 : 2 & = & 3 \text{ (0)} \\
 3 : 2 & = & 1 \text{ (1)} \\
 1 : 2 & = & 0 \text{ (1)}
 \end{array}$$

Результат: $105_{10} = 1101001_2$.

Переведення числа 105 до шіснадцяткової системи числення матиме вигляд:

$$105 : 16 = 6 \text{ (9)}$$

Результат: $105_{10} = 69_{16}$.

Виконання побітових логічних операцій:

1) інвертувати усі біти числа

$$0110 \ 1001_2 = 1001 \ 0110_2 = 1 \cdot 2^7 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 128 + 16 + 4 + 2 = 150;$$

Проте, якщо число займає не один, а більше байтів, наприклад, два або чотири, то інвертуватимуться і старші біти числа з 0 на 1. Для двобайтного числа 105 (0000 0000 1001 0110) інвертованим буде 1111 1111 0110 1001 = 65 430.

Це є логічним, оскільки $65\ 535 - 105 = 65\ 430$;

2) визначити (вибрати окремо) значення 3-го біта числа:

$$110 \ 1001_2 \rightarrow 1;$$

3) встановити (задати значення 1) для 4-го біта числа:

$$110 \ 1001_2 = 111 \ 1001 = 2^6 + 2^5 + 2^4 + 2^3 + 2^0 = 64 + 32 + 16 + 8 + 1 = 121;$$

4) скинути (встановити в 0) 0-й біт числа:

$$110 \ 1001_2 = 110 \ 1000_2 = 2^6 + 2^5 + 2^3 = 64 + 32 + 8 = 104;$$

5) інвертувати 2-й біт числа:

$$110 \ 1001_2 = 110 \ 1101_2 = 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^0 = 64 + 32 + 8 + 4 + 1 = 109;$$

6) логічний побітовий зсув на 2 біти праворуч:

$$1101001_2 \gg 2 = 11010_2 = 2^4 + 2^3 + 2^1 = 16 + 8 + 2 = 26 \ (105/2^2 = 105/4 = 26).$$

Програмний код:

```

#include <iostream>
using namespace std;
// Функція для переведення числа у двійкову систему числення
void bin (unsigned short dec)
{ int v, i, kol; // Вага і номер формованого розряду, а також кількість розрядів
  if (dec > 255)
    { v = 65536; kol=16; } // kol - кількість розрядів

```

```

else { v=256; kol=8; } // v - вага старшого розряду
for (i = 1; i <= kol; i++)
{ v = v / 2; // Вага наступного розряду удвічі менше
  if (i % 4 == 1) printf(" ");
  if (dec >= v)
  { cout<<"1";
    dec -= v;
  }
  else printf("0");
}
}
int main()
{
  unsigned short x, xinv, x3, x4_1, x0_0, xinv_2, x_r2;
  cout << "Input number :"; cin >> x;
  bin(x); // Виклик функції для переведення числа у двійкову систему числення.
  cout << "\n\nBinary operations : \n";
  xinv = ~x; // Побітова інверсія x
  cout << "Invert number (size - 2 byte) : " << xinv << " = "; bin(xinv);
  if (x<255) // Якщо число займає менше 2 байтів, тобто його значення менше 255,
  { // позбавитись від старшого байту за рахунок побітового множення на 0xFF
    xinv = xinv & 255;
    cout << "\nInvert number (size - 1 byte) : " << xinv << " = ";
    bin(xinv);
  }
  // Наступні 3 команди визначають 3 біт числа за рахунок зсуву усіх бітів праворуч і
  // ліворуч від 3-го
  // k - кількість бітів праворуч від 3-го (нумеруються біти, починаючи з 0)
  int k = 16-3-1;
  x3 = x << k; x3 = x3 >> (k+3);
  cout << endl<< "3 bit = " << x3 << endl;
  // Встановлення 4-го біта числа в 1 за рахунок логічного побітового додавання з
  // числом 0001 0000
  x4_1 = x | 0x10; // або x/16
  cout << "Number with 4bit=1 : " << x4_1 << " = "; bin(x4_1);
  // Встановлення 0-го біта в 0 логічним побітовим множенням з 1111 1111 1111 1110
  x0_0 = x & 0xFFFE;
  cout << "\nNumber with 0bit=0 : " << x0_0 << " = "; bin(x0_0);
  // Інверсія 2-го біта числа за рахунок логічного виключного АБО з 0000 0100
  xinv_2= x^(1<<2); // або x^4
  cout << "\nNumber after invert 2bit : " << xinv_2 << " = "; bin(xinv_2);
  x_r2=x>>2; // Побітовий зсув праворуч на 2-го біти
  cout << "\nNumber after >>2 : " << x_r2 << " = "; bin(x_r2);
  cout << endl;
  system("pause>>void");
  return 0;
}

```


Результати виконання програми:

```
Input number :105
0110 1001

Binary operations :
Invert number (2 byte) : 65430 = 1111 1111 1001 0110
Invert number (1 byte) : 150 = 1001 0110
3 bit = 1
Number with 4bit=1 : 121 = 0111 1001
Number with 0bit=0 : 104 = 0110 1000
Number after invert 2bit : 109 = 0110 1101
Number after >>2 : 26 = 0001 1010
```

4.2.4 Пріоритет логічних операцій

Обчислення в логічних виразах виконуються зліва направо у відповідності з таким *пріоритетом* операцій :

- 1) ~, !;
- 2) >>, <<;
- 3) <, <=, >, >=;
- 4) ==, !=;
- 5) & (побітове «І»);
- 6) ^ (побітове «виключне АБО»);
- 7) | (побітове «АБО»);
- 8) && («І»);
- 9) || («АБО»).

Вирази у круглих дужках виконуються першочергово.

4.2.5 Пріоритети операцій і порядок обчислень

З урахуванням вищенаведеного пріоритету логічних операцій та пріоритету арифметичних операцій, наведеного в підрозд. 3.5, наведемо загальний пріоритет для всіх операцій мови C в табл. 4.3. Операції з вищими пріоритетами обчислюються першими.

Таблиця 4.3

Пріоритет і порядок виконання операцій

Пріоритет	Знак операції	Тип операції	Порядок виконання
1	::, (), [], ., ->	вираз	зліва направо
2	~, !, *, &, ++, --, sizeof, (<min>)	унарні	справа наліво
3	*, /, %	мультиплікативні	зліва направо
4	+, -	адитивні	
5	<<, >>	зсув	
6	<, >, <=, >=	відношень	
7	==, !=	відношень (порівнянь)	

Закінчення табл. 4.3

Пріоритет	Знак операції	Тип операції	Порядок виконання
8	&	побітове «І»	зліва направо
9	^	побітове «виключне АБО»	
10		побітове «АБО»	
11	&&	логічне «І»	
12		логічне «АБО»	
13	? :	умовна	справа наліво
14	=, *=, /=, % =, +=, -=, &=, =, >>=, <<=, ^=	просте і складені присвоювання	
15	,	послідовне обчислення	зліва направо

4.2.6 Умовний оператор if

Оператор **if** має дві форми: скорочену та повну.

Скорочена форма має вигляд:

if (<умова>) <оператор>;

Якщо значення логічного виразу в умові є ненульове, тобто умова є істинна, то виконується *оператор* чи група операторів в операторних дужках {}, інакше відбудеться перехід на наступний оператор.

Повна форма цього оператора:

if (<умова>) <оператор1>;
else <оператор2>;

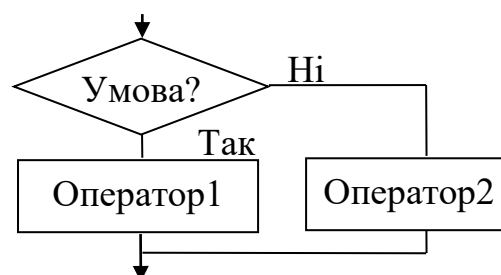
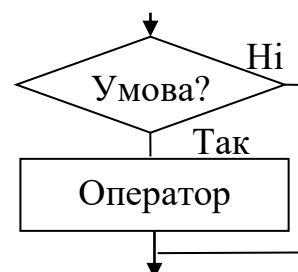
Якщо значення логічного виразу в умові є ненульове, тобто умова є істинна, то виконується *оператор1* чи група операторів в операторних дужках {}, інакше – виконується *оператор2*, після чого відбудеться перехід на наступний оператор. Зауважимо, що обчислюється лише один з операторів, а не обидва.

Наприклад, обчислення значення виразу

$$y = \begin{cases} 1 + b^x & \text{за } x = b; \\ \frac{x + b}{b - x} & \text{за } x \neq b \end{cases}$$

можна реалізувати чи то двома операторами **if** скороченої форми, чи одним оператором **if** повної форми:

- 1) **if** (x == b) y = 1 + pow(b, x);
 if (x != b) y = (x + b)/(b - x);
- 2) **if** (x == b) y = 1 + pow(b, x); **else** y = (x + b)/(b - x);



Зауважимо, що умову ($x \neq 0$) можна записати чи то як ($x \neq 0$), чи як (x). І, навпаки, для того щоб перевірити, чи дорівнює нулю певне число x , умова може мати вид ($x==0$) чи ($!x$).

Окрім того, слід розрізняти вирази в умовах типу

```
if (x == 5) . . .;
```

та

```
if (x = 5) . . .;
```

Обидва ці оператори є працездатними, оскільки в C++ будь-який вираз, який має певне числове значення, може використовуватися в умовних операторах. Але, якщо у першому випадку значення логічного виразу може бути як істинним, так і хибним, залежно від значення змінної x , то у другому випадку значення логічного виразу завжди є істинне, позаяк в результаті присвоювання значення $x = 5$ – ненульове.

Наведемо кілька прикладів використання умовного оператора `if`.

1) Обчислити частку від ділення двох дійсних чисел, якщо це є можливо.

```
if (y != 0)    // Якщо знаменник не 0, можна обчислити
    r = x / y;
```

Це приклад скороченої форми умовного оператора (без гілки «інакше»).

2) Обчислити квадратний корінь числа, якщо це є можливо.

```
if (x >= 0)    // Якщо під коренем додатне число, можна обчислити,
    y = sqrt(x);
else          // інакше – виводимо повідомлення, що обчислити неможливо
    cout<<"Обчислити неможливо!"<<endl;
```

Це приклад повної форми умовного оператора (з гілкою «інакше»).

3) Якщо змінні x та y є однаковими, обчислити їхній добуток, інакше – обчислити модуль їх різниці.

```
if (x == y) r = x * y;
else r = fabs(x - y);
```

4) Якщо модуль змінних x та y є менше за 10, збільшити їх удвічі; інакше – зменшити їх удвічі.

```
if (fabs(x)<10 && fabs(y)<10)
{
    x=x*2;
    y=y*2;
}
else
{
    x=x/2;
    y=y/2;
}
```

У цьому прикладі, оскільки за істинності умови має виконуватись набір з двох команд, ці команди слід записати в операторних дужках `{}`.

Команди в операторних дужках `{}` називаються *блоком команд*.

Умовні оператори можуть бути необмежено вкладеними один в одного. У деяких випадках складно розібратися, в якій послідовності виконуються такі вкладені оператори. Вельми часто зустрічається використання конструкції

if - else - if. Наприклад, щоб визначити знак змінної x (вона може бути додатною, від'ємною чи нулем), можна використовувати три простих незалежних команди **if**:

```
if (x > 0)   cout<<"positive";
if (x < 0)   cout<<"negative";
if (x == 0)  cout<<"zero";
```

чи оператор **if - else** вкладеної конструкції:

```
if (x > 0) cout<<"positive";
else
    if (x < 0) cout<<"negative";
    else cout<<"zero";
```

Тут, якщо спрацював перший **else**, це означає, що x є від'ємним числом чи нульовим. Тому слід перевірити обидві гілки за допомогою ще однієї команди **if**.

Обчислення функції

$$y = \begin{cases} \sin(x + a) & \text{за } x \leq -2; \\ \cos(x - \pi)^2 & \text{за } -2 < x \leq 3; \\ \operatorname{arctg}^2 \frac{x}{a} & \text{за } x > 3, a \neq 0; \\ 1 & \text{за } x > 3, a = 0, \end{cases}$$

аналогічно до попереднього прикладу, можна реалізувати чи то за допомогою чотирьох операторів **if** скороченої форми, чи оператором **if - else** вкладеної конструкції:

- 1) **if** (x <= -2) y = sin(x + a);
if (x > -2 && x <= 3) y = cos(pow(x - M_PI, 2));
if (x > 3 && a != 0) y = pow(atan(x / a), 2);
if (x > 3 && a == 0) y = 1;
- 2) **if** (x <= -2) y = sin(x + a);
else
if (x <= 3) y = cos(pow(x - M_PI, 2));
else
if (a != 0) y = pow(atan(x / a), 2);
else y = 1;

У конструкціях **if - else - if** умови операторів **if** перевіряються зверху донизу. Тільки-но якась з умов набуває ненульового значення, одразу виконуватиметься оператор, який слідує за цією умовою, а останню частину конструкції буде проігноровано.

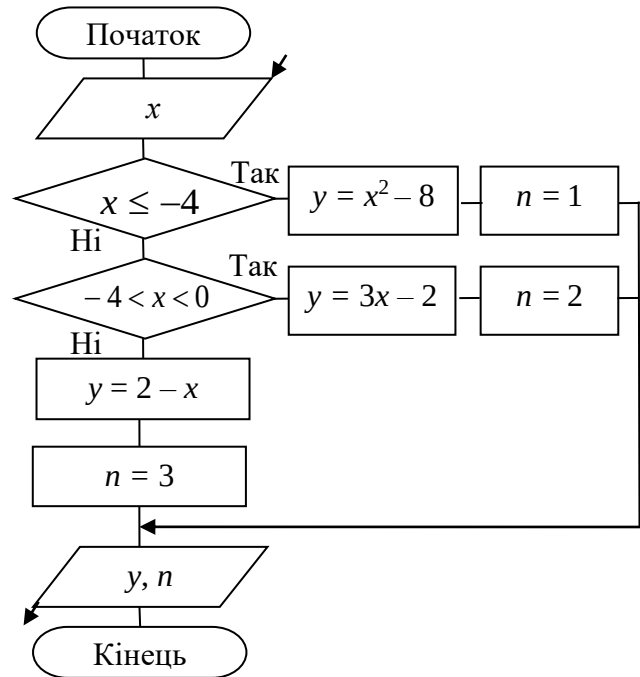
Слід записувати подібні вкладені оператори максимально наочно, використовуючи відступи з пропусків, уникати заплутаних складових і великого рівня вкладеності.

Приклади створювання проектів програм з розгалуженою структурою, організованих за допомогою оператора if

Приклад 4.12. Ввести x та обчислити $y = \begin{cases} x^2 - 8 & \text{за } x \leq -4; \\ 3x - 2 & \text{за } -4 < x < 0; \\ 2 - x & \text{за } x \geq 0. \end{cases}$

Текст програми та схема алгоритму:

```
#include <iostream>
using namespace std;
int main ()
{ setlocale(0, ".1251");
  double y, x; int n;
  cout<<"Введіть значення x=";
  cin>>x;
  if (x <= -4) { y=x*x-8; n=1;}
  else
    if (-4<x && x<0) { y=3*x-2; n=2;}
    else { y=2-x; n=3;}
  cout<< "Результат y="<< y << endl;
  cout<< "Виконано умову "<< n << endl;
  system ("pause>>void");
  return 0;
}
```



Результати роботи:

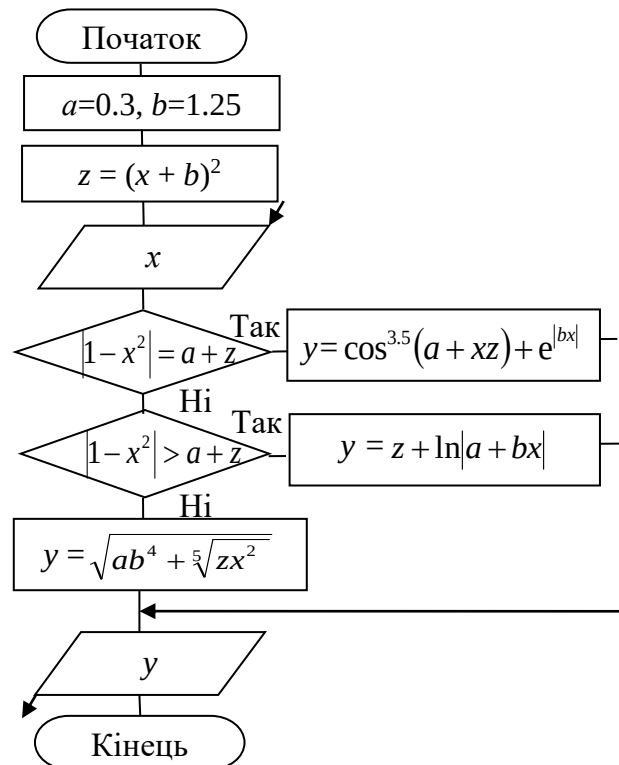
Введіть значення $x=0.4$
Результат $y=1.6$
Виконано умову 3

Приклад 4.13. Ввести x , задати $a = 0.3$; $b = 1.25$; $z = (x + b)^2$ та обчислити

$$y = \begin{cases} \cos^{3.5}(a + xz) + e^{|bx|} & \text{за } |1 - x^2| = a + z; \\ z + \ln|a + bx| & \text{за } |1 - x^2| > a + z; \\ \sqrt{ab^4 + \sqrt[5]{zx^2}} & \text{за } |1 - x^2| < a + z, \end{cases}$$

Текст програми та схема алгоритму:

```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{
  setlocale(0, ".1251");
  double a=0.3, b=1.25, x, y, z;
  cout<<"Введіть значення x= ";
  cin>>x;
  z = pow(x + b, 2);
```



```

if (fabs(1-x*x) == a+z) y = pow(cos(a+x*z),3.5) + exp(fabs(b*x));
else
    if (fabs(1-x*x) > a+z) y = z+log(fabs(a+b*x));
    else y = sqrt(a * pow(b,4) + pow(z*x*x,1./5));
cout<<"\nРезультат y = "<< y <<endl;
system ("pause>>void");
return 0;
}

```

Результати роботи:

Введіть значення $x = 2.45$

Результат $y = 1.77414$

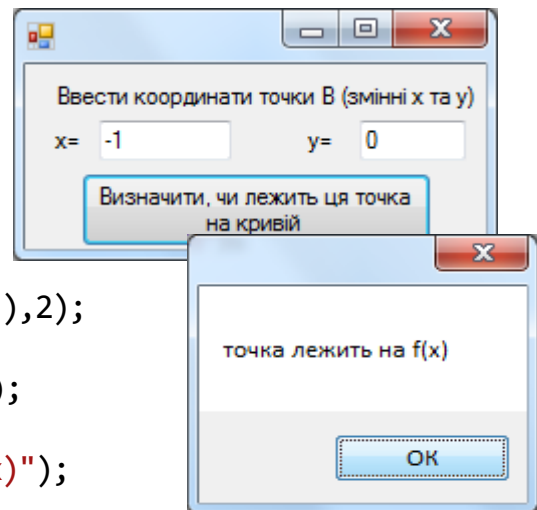
Приклад 4.14. Ввести координати точки B (змінні x та y) та визначити, чи лежить ця точка на кривій $f(x) = \begin{cases} -e^x + x^4 & \text{за } |x| > 1; \\ \arccos^2 x^4 & \text{за } |x| \leq 1. \end{cases}$ Похибку задати $\text{eps} = 10^{-3}$ (тобто $|f(x) - y| < \text{eps}$). Відповідь вивести у вигляді повідомлення.

Текст програми:

```

private: System::Void button1_Click(System::Object^sender, System::EventArgs^e)
{ double x, y, fx;
  const double eps = 1e-3;
  x = Convert::ToDouble(textBox1->Text);
  y = Convert::ToDouble(textBox2->Text);
  if (Math::Abs(x) > 1)
    fx = -Math::Exp(x) + Math::Pow(x, 4);
  else
    fx = Math::Pow(Math::Acos(Math::Pow(x,4)),2);
  if (Math::Abs(fx - y) < eps)
    MessageBox::Show("точка лежить на f(x)");
  else
    MessageBox::Show("точка не лежить на f(x)");
}

```



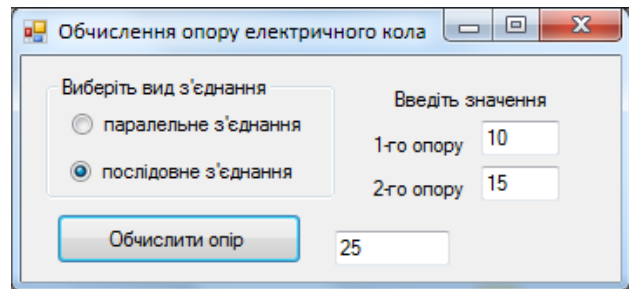
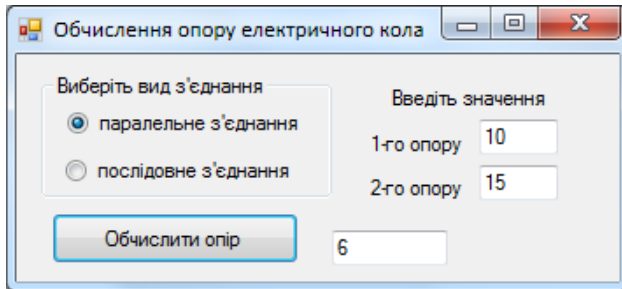
Приклад 4.15. Створити програму для обчислення опору електричного кола, який складається з двох з'єднаних опорів, залежно від виду їхнього з'єднання: паралельне або послідовне.

Тобто вдосконалимо розв'язок прикладу 4.4 з підрозд 4.1 за рахунок можливості вибору виду з'єднаних опорів. Для цього використаємо елемент `groupBox`, на якому розмістимо два елементи `radioButton`. Для `radioButton1` слід встановити значення `Checked` в `True`. Перемикачі `radioButton` дозволяють «увімкнути» лише одну з усіх радіокнопок , при цьому решта кнопок автоматично вимикається.

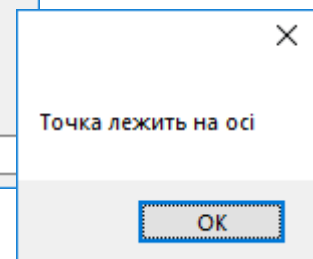
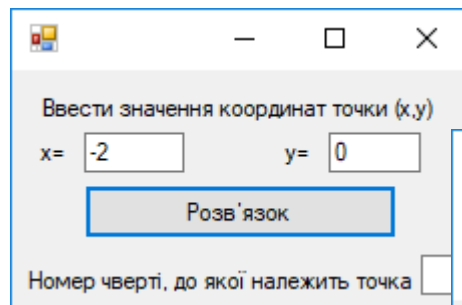
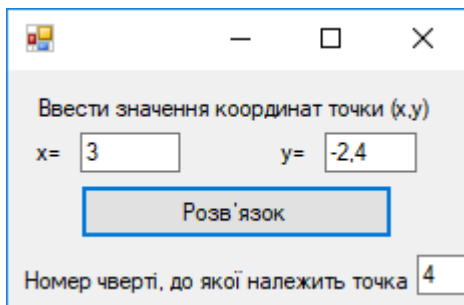
Програмне опрацювання вибору кнопки можна реалізувати оператором `if-else`.

Текст програми:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    double r1, r2, R;
    r1 = Convert::ToDouble(textBox1->Text); // значення 1-го опору r1
    r2 = Convert::ToDouble(textBox2->Text); // значення 2-го опору r2
    if (radioButton1->Checked) R = r1*r2/(r1+r2);
    else R = r1+r2;
    textBox3->Text = Convert::ToString(R);
}
```



Приклад 4.16. Ввести значення координат (x, y) точки площини і визначити, до якої чверті площини належить ця точка.



Текст програми:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    double x, y; int nomer;
    x = Convert::ToDouble(textBox1->Text);
    y = Convert::ToDouble(textBox2->Text);
    if ((x==0) || (y==0))
    {
        textBox3->Clear();
        MessageBox::Show("Точка лежить на осі");
        return;
    }
    else
    {
        if (x>0 && y>0) nomer = 1;
        else
        {
            if (x<0 && y>0) nomer = 2;
            else
            {
                if (x<0 && y<0) nomer = 3;
                else nomer = 4;
            }
        }
        textBox3->Text = nomer.ToString();
    }
}
```

```

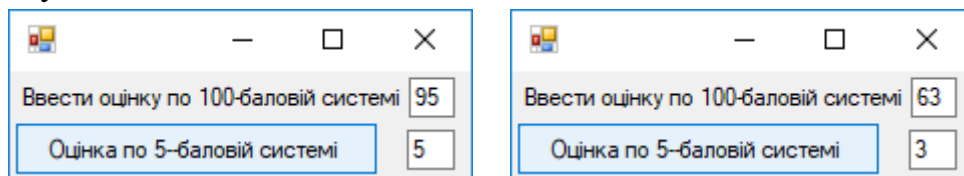
    if (x<0 && y<0) nomer=3;
    else
        nomer = 4;
    textBox3->Text = Convert::ToString(nomer);
}

```

Метод `MessageBox::Show()` виводить на екран діалогове вікно з текстом відповідного повідомлення та однією командною кнопкою *ОК*, при натисканні на яку вікно повідомлення зникає.

Приклад 4.17. Перевести оцінку по 100-баловій системі до 5-бальної за алгоритмом: оцінка менша за 60 відповідає незадовільній оцінці «2»; в межах від 60 до 73 – оцінці «3»; від 74 до 89 – «4»; розпочинаючи з 90 і до 100 – «5».

Зауважимо, що в різних вузах можливі незначні відмінності від наведеного алгоритму.



Текст програми:

```

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    int m100, m5;
    m100=Convert::ToInt32(textBox1->Text);
    if (m100>=90) m5=5;
    if (m100>=74 && m100<90) m5=4;
    if (m100>=60 && m100<74) m5=3;
    if (m100<60) m5=2;
    textBox2->Text = Convert::ToString(m5);
}

```

Аналогічна програма із вкладеною конструкцією `if – else – if`:

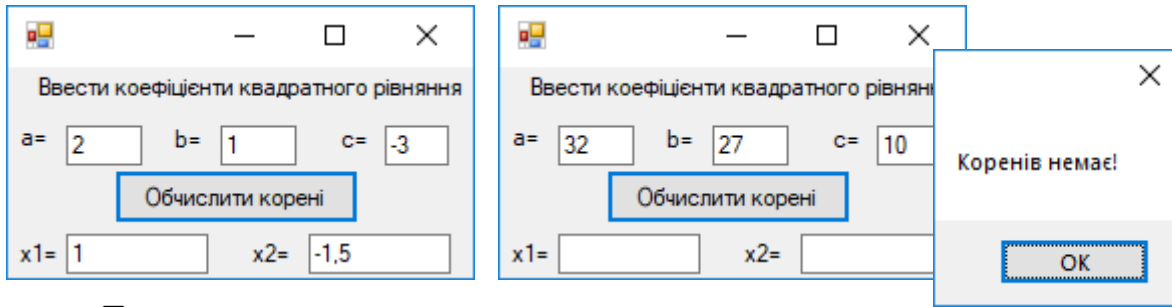
```

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    int m100, m5;
    m100=Convert::ToInt32(textBox1->Text);
    if (m100>=90) m5=5;
    else
        if (m100>=74 && m100<90) m5=4;
        else
            if (m100>=60 && m100<74) m5=3;
            else m5=2;
    textBox2->Text = Convert::ToString(m5);
}

```

У цьому варіанті програми немає потреби перевіряти праву межу інтервалів.

Приклад 4.18. Створити програмний проект для обчислення коренів квадратного рівняння $ax^2 + bx + c = 0$. Передбачити перевірку наявності можливих коренів, тобто перевірку, чи є дискримінант додатним.



Текст програми:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{ double a, b, c, D, x1, x2;
  a=Convert::ToInt32(textBox1->Text);
  b=Convert::ToInt32(textBox2->Text);
  c=Convert::ToInt32(textBox3->Text);
  D=b*b-4*a*c;
  if (D<0)
  { textBox4->Clear(); textBox5->Clear();
    MessageBox::Show("Коренів немає!");
  }
  else
  { x1=(-b + Math::Sqrt(D))/(2*a);
    x2=(-b - Math::Sqrt(D))/(2*a);
    textBox4->Text = Convert::ToString(x1);
    textBox5->Text = Convert::ToString(x2);
  }
}
```

Приклад 4.19. Обчислити силу струму I на ділянці електричного ланцюга довжиною L за формулою

$$I = \begin{cases} \sin^2(LT) - \cos^2\left(\frac{L}{T}\right) & \text{за } 0 \leq L \leq \frac{1}{3}; \\ \sin^2\left(\frac{T}{L}\right) + \cos^2\left(\frac{1}{TL}\right) & \text{за } \frac{1}{3} < L \leq \frac{2}{3}; \\ e^{LT} \left(\sin^2\left(\frac{1}{TL}\right) - \cos^2(LT) \right) & \text{за } \frac{2}{3} < L \leq 1. \end{cases}$$

Значення періоду T та довжини L ввести.

Текст програми та скріншот вікна онлайн-компілятора <http://cpp.sh>:

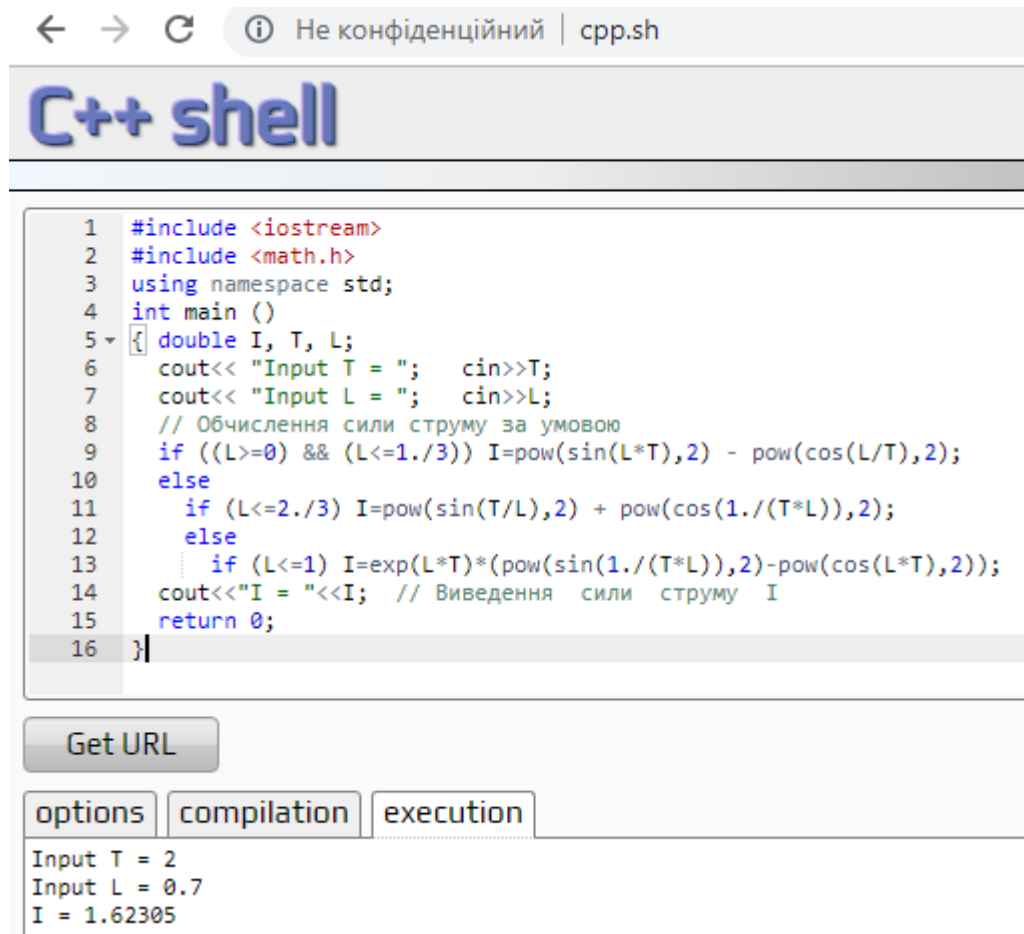
```
#include <iostream>
#include <math.h>
using namespace std;
int main ()
{ double I, T, L;
```



```

cout<< "Input T = ";   cin>>T;
cout<< "Input L = ";   cin>>L;
// Обчислення сили струму за умовою
if ((L>=0) && (L<=1./3)) I=pow(sin(L*T),2) - pow(cos(L/T),2);
else
    if (L<=2./3) I=pow(sin(T/L),2) + pow(cos(1./(T*L)),2);
    else
        if (L<=1) I=exp(L*T)*(pow(sin(1./(T*L)),2)-pow(cos(L*T),2));
cout<<"I = "<<I; // Виведення сили струму I
return 0;
}

```



Приклад 4.20. Ввести значення року та перевірити, чи є він високосним.

Текст програми та скріншоти результатів роботи програми в консолі Visual C++:

```

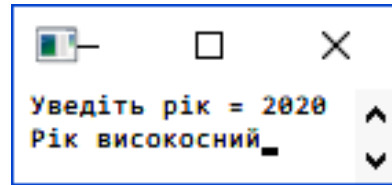
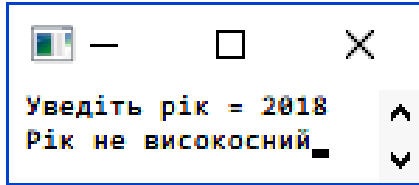
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
    setlocale(0, ".1251");
    system("color F0"); // Встановити білий фон і чорний текст
    int year, r;

```

```

cout<< " Уведіть рік = " ; cin>> year;
r = year % 4;           // Остача від ділення year на 4
if (r) cout<< " Рік не високосний";
else cout<< " Рік високосний";
_getch();               // або system ("pause">>void");
return 0;
}

```



Приклад 4.21. Ввести ненульове ціле число і визначити його знак (вивести «Додатне» чи «Від’ємне»).

Текст програми:

```

#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
    setlocale(0, ".1251");
    int x;
    cout << "Ввести ненульове число:";    cin >> x;
    if (x>0) cout << "Додатне" << endl;
    else cout << "Від’ємне" << endl;
    _getch();
    return 0;
}

```

Результати виконання програми:

Ввести число: 5
Додатне

Ввести число: -15
Від’ємне

Приклад 4.22. Написати програму для перевірки знання року заснування Одеси. У разі помилкової відповіді користувача програма має вивести правильну відповідь.

Текст програми:

```

#include <iostream>
#include <conio.h>
using namespace std;
int main()
{
    setlocale(0, ".1251");
    int year;

```

```

cout << "Введіть рік заснування Одеси= " ;
cin >> year ;
if (year==1794) cout << "Відповідь є правильна "<<endl;
else cout<<"Ви помилились. Одесу було засновано 1794 року";
_getch();
return 0;
}

```

Результати виконання програми:

Введіть рік заснування Одеси=1790
Ви помилились. Одесу було засновано 1794 року

Введіть рік заснування Одеси=1794
Відповідь є правильна

Приклад 4.23. Ввести число й обчислити, якщо це можливо, значення виразу $y = \frac{3}{\sqrt{x-7}-2}$.

Розв'язок. Обчислити значення цього виразу можливо, якщо, по-перше, під коренем розташовано невід'ємне число і, по-друге, знаменник не дорівнює 0, тобто якщо виконуються дві умови: $x-7 \geq 0$ та $\sqrt{x-7}-2 \neq 0$.

Текст програми:

```

#include <iostream>
#include <math.h>
#include <conio.h>
using namespace std;
int main()
{
    setlocale(0, ".1251");
    float x, y;
    cout << "Введіть число: " ;
    cin >> x;
    if (x-7 >= 0 && sqrt(x-7)-2 != 0)
    {
        y = 3/(sqrt(x-7)-2);
        cout << "y= " << y << endl;
    }
    else cout<<"Обчислити неможливо"<<endl;
    _getch();
    return 0;
}

```

Результати виконання програми:

Введіть число: 15
y=3.62132

Введіть число: 5
Обчислити неможливо

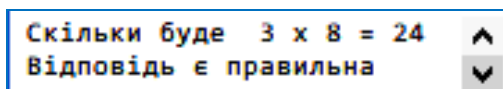
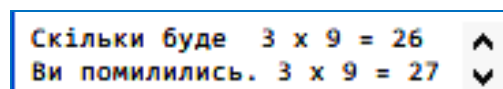
Приклад 4.24. Написати програму для перевірки знання таблиці множення, тобто множення двох випадкових одноцифрових чисел.

Розв'язок. Після кожної відповіді відбувається перевірка її правильності та виводиться відповідне повідомлення.

Текст програми та скріншоти результатів роботи програми в консолі Visual C++:

```
#include <iostream>
#include <conio.h>
#include <stdlib.h>
#include <time.h>

using namespace std;
int main()
{
    setlocale(0, ".1251");
    system("color F0"); // Встановити білий фон і чорний текст
    int x,y,p,ans;       // Співмножники, добуток та відповідь
    time_t t;           // Поточний час для ініціалізації генератора випадкових чисел
    // Ініціалізація генератора випадкових чисел (див. докладніше підрозд. 3.6)
    srand((unsigned) time(&t));
    x=rand()%10+1;      y=rand()%10+1;
    p=x*y;
    cout << " Скільки буде " << x << " x " << y << " = ";
    cin >> ans ;
    if(p==ans)cout<<" Відповідь є правильна"<<endl;
    else cout << " Ви помилились. " << x << " x " << y << " = " << p;
    _getch(); // або system ("pause">>void");
    return 0;
}
```

4.2.7 Тернарна умовна операція ?:

У мові C++ є одна тернарна операція, яка має три операнди, це – умовна операція, котра має такий формат:

<умова> ? <операнд1> : <операнд2>;

Якщо *умова* має ненульове значення, результатом буде значення *операнда1*, інакше – значення *операнда2*. Зауважимо, що обчислюється лише один з операндів, а не обидва. Наприклад:

j = (i < 0) ? (-i) : (i);

У результаті *j* набуде абсолютного значення *i*, оскільки, якщо *i* є менше за 0, то *j* присвоїться *-i*, а якщо *i* є більше чи дорівнює нулю, то *j* присвоїться *i*.

Для визначення більшого з двох чисел – *x* та *y* – слід записати оператор:

max = (x > y) ? x : y;

Для знакопочередного ряду вираз $(-1)^i$ можна записати без використання математичної функції, суттєво зекономивши пам'ять і час виконання, за допомогою умовної операції: $(i\%2) ? -1 : 1$

Наприклад, організувати почергове змінювання тексту на кнопці при клацанні по ній можна лише однією командою:

```
button1->Text = (button1->Text == "on") ? "off" : "on";
```

Тип результату залежить від типів *операнда1* і *операнда2*:

✓ якщо *операнд1* чи *операнд2* має цілий чи дійсний тип (зауважимо, що їхні типи можуть відрізнятися), то виконуються звичайні арифметичні перетворення. Типом результату є тип операнда після перетворення;

✓ якщо *операнд1* чи *операнд2* має один і той самий тип структури, об'єднання чи вказівника, то тип результату буде тим самим типом структури, об'єднання чи вказівника;

✓ якщо обидва операнди мають тип `void`, то результат матиме тип `void`;

✓ якщо один операнд є вказівником на об'єкт будь-якого типу, а другий операнд є вказівником на `void`, то вказівник на об'єкт перетворюється на вказівник на `void`, котрий і буде типом результату;

✓ якщо один операнд є вказівником, а другий – константним виразом зі значенням 0, то типом результату буде тип вказівника.

Умовна операція `?:` за своєю дією є схожою до оператора `if-else`, однак слід зважати, що вона використовується лише у виразах і передає значення одного з операндів. Наприклад, оператор

```
if ( z ) res = x; else res = y;
```

можна записати за допомогою умовної операції так

```
z ? (res = x) : (res = y) ;
```

або ясніше для сприйняття

```
res = z ? x : y ;
```

Окрім того, зауважимо, що таку конструкцію можна записати і за допомогою логічних операцій `&&` та `||`

```
z && (res = x);
```

```
z || (res = y);
```

але таку заміну більшість людей сприймають важко і неоднозначно.

Умовні операції можна вкладати одна в одну. Наприклад, залежно від значення числа можна формувати коректну кінцівку рядка:

```
int k;
cout<<"Введіть значення кількості комп'ютерів - "; cin>> k;
int n=(k>20)? k%10 : k;
cout<<"В класі є " << k << " комп'ютер"
<< ((!n || n>4) ? "ів." : (n>1) ? "и." : ".")<<endl;
```

```
Введіть значення кількості комп'ютерів - 3
В класі є 3 комп'ютери.
```

```
Введіть значення кількості комп'ютерів - 15
В класі є 15 комп'ютерів.
```

```
Введіть значення кількості комп'ютерів - 21
В класі є 21 комп'ютер.
```

Розглянемо на прикладі, як застосовувати умовну операцію `?:`, і побачимо, що її результат є ясніший для сприйняття, аніж рівноцінний оператор `if-else`.

Приклад 4.25. Ввести два числа і вибрати з них більше.

Текст програмного коду:

```
#include <iostream>
using namespace std;
int main()
{
    int x, y, z;           // Оголошуємо три змінні
    cout << "Enter two numbers:\n";
    cin >> x >> y;        // Вводимо з клавіатури два числа
    if (x>y) z=x;          // Якщо перше число є більше за друге, присвоюємо його змінній z,
    else z=y;              // інакше – присвоюємо змінній z друге число
    cout << "\n \nResalt 1: " << z; // Виводимо z
    z = (x > y) ? x : y;    // Тернарна операція
    cout << "\nResalt 2: " << z << "\n";
    return 0;
}
```

Результати роботи консольної програми:

Enter two numbers:

15 8

Resalt 1: 15

Resalt 2: 15

Наведену у програмі тернарну операцію читають як: «Якщо `x` більше за `y`, повернути значення `x`. Інакше – повернути значення `y`». Значення, яке повертається, присвоюється змінній `z` і виводиться на екран. Вочевидь, що тут умовна операція є скороченим еквівалентом команди `if-else`.

Однак бувають ситуації, коли операцію `?:` не можна замінити конструкцією `if-then-else`, наприклад, якщо організувати попередню програму так:

```
#include <iostream>
using namespace std;
int main()
{
    int x, y;              // Оголошуємо дві змінні
    cout << "Enter two numbers:\n";
    cin >> x >> y;        // Вводимо з клавіатури два числа
    int z = (x > y) ? x : y; // Тернарна операція
    cout << "\n max = " << z << "\n";
    return 0;
}
```

Тут змінна `z` ініціалізується на момент оголошення результатом роботи тернарної операції. Подібного ефекту не вдалося б досягти простим привласненням в тому чи іншому випадку.

4.2.8 Оператор вибору варіантів switch

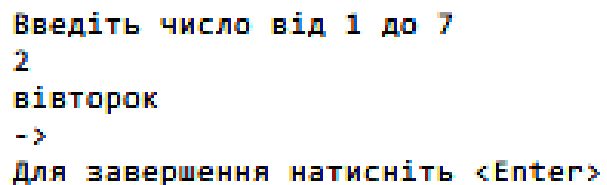
Формат оператора вибору варіантів `switch`:

```
switch (<вираз>)  
{ case <значення-мітка_1> : <послідовність операторів>; break;  
  . . . . .  
  case <значення-мітка_n> : <послідовність операторів>; break;  
  [ default: <послідовність операторів>; ]  
}
```

Спочатку обчислюється *вираз* у дужках. *Вираз* повинен мати цілий чи то символьний тип. Значення *виразу* порівнюється зі значеннями *міток* після ключових слів `case`. Якщо значення *виразу* збіглося зі значенням якоїсь *мітки*, то виконується відповідна *послідовність операторів*, позначена цією *міткою* і записана після двокрапки, доки не зустрінеться оператор `break`. Якщо значення *виразу* не збіглося з жодною *міткою*, то виконуються оператори, які слідує за ключовим словом `default`. Мітка `default` є необов'язковою конструкцією оператора `switch`, на що вказують квадратні дужки `[]` у форматі. Оператор `break` здійснює вихід із `switch`. Якщо оператор `break` є відсутній наприкінці операторів відповідного `case`, то буде по чергову виконано всі оператори до наступного `break` чи то до кінця `switch` для всіх гілок `case` незалежно від значення їхніх міток.

Розглянемо роботу оператора `switch` на прикладі програми, яка за числом виводить назву дня тижня.

```
#include <iostream>  
#include <conio.h>  
using namespace std;  
int main()  
{ setlocale(0, ".1251");  
  system("color F0");  
  int n;  
  puts(" Введіть число від 1 до 7");  
  scanf_s("%i",&n);  
  switch (n)  
  { case 1: puts(" понеділок"); break;  
    case 2: puts(" вівторок"); break;  
    case 3: puts(" середа"); break;  
    case 4: puts(" четвер"); break;  
    case 5: puts(" п'ятниця"); break;  
    case 6: puts(" субота"); break;  
    case 7: puts(" неділя"); break;  
    default: puts(" Помилка! Введіть число від 1 до 7");  
  }  
  puts(" ->\n Для завершення натисніть <Enter> \n");  
  _getch();  
  return 0;  
}
```



```
Введіть число від 1 до 7  
2  
вівторок  
->  
Для завершення натисніть <Enter>
```

Наведемо ще один приклад програми з використанням оператора `switch` для реалізації простого калькулятора в консольному режимі.

```
#include <iostream>
using namespace std;
int main()
{ setlocale(0, ".1251");
  double a,b,res;  char op;
  cout << "\n Введіть перший операнд "; cin >> a;
  cout << "\n Введіть знак операції "; cin >> op;
  cout << "\n Введіть другий операнд "; cin >> b;
  bool f=true;
  switch (op)
  { case '+' : res = a+b; break;
    case '-' : res = a-b; break;
    case '*' : res = a*b; break;
    case '/' : res = a/b; break;
    default: cout << "\n Невідома операція!\n "; f=false;
  }
  if (f) cout << "\n Результат : " << res<<endl;
  system("pause>>void");
  return 0;
}
```

```
Введіть перший операнд 0.25
Введіть знак операції *
Введіть другий операнд 50
Результат : 12.5
```

```
Введіть перший операнд 89
Введіть знак операції ^
Введіть другий операнд 10
Невідома операція!
```

Приклади створювання проектів програм з розгалуженою структурою, організованих за допомогою оператора `switch`

Приклад 4.26. Обчислити значення функції
$$L = \begin{cases} x^{2+a} + 1 & \text{за } k = 1; \\ (x+1)/a & \text{за } k = 2; \\ e^{x-a} - a^x & \text{за } k = 3; \\ \lg|x-a| & \text{за } k = 4 \end{cases}$$

для всіх значень параметра k .

Розв'язок. Спочатку наведемо текст програми у консолі, а тоді покажемо програмний проект з формою, де виведення всіх чотирьох значень функції L організуємо до елемента `richTextBox` (багаторядкове текстове вікно для введення чи виведення значень даних програми). Схема алгоритму не залежить від способу реалізації програмного коду.

Текст консольної програми у Visual C++:

```
#include <iostream>
#include <math.h>
```



```

using namespace std;
int main()
{
    setlocale(0, ".1251");
    double x, a, L;    int k = 1;
    cout<<"Введіть значення a= ";    cin>> a;
    cout<<"Введіть значення x= ";    cin>> x;
    cout<< "\nРезультати:\n";
M1:
    switch (k)
    { case 1: { L = pow(x,2+a) + 1; break; }
      case 2: { L = (x+1)/a; break; }
      case 3: { L = exp(x-a) - pow(a,x); break; }
      case 4: { L = log10(fabs(x-a)); break; }
    }
    cout<< "L" << k << " = " << L << endl;
    k++;
    if (k <= 4) goto M1;
    system("pause>>void");
    return 0;
}

```

```

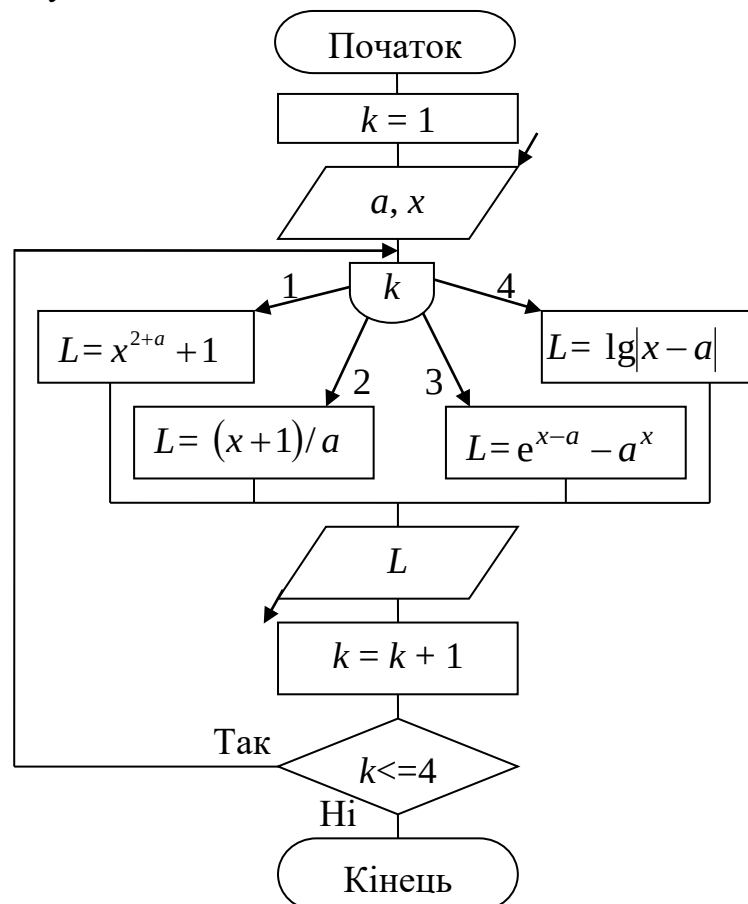
Введіть значення a= 0.8
Введіть значення x= 10.5

Результати:
L1 = 724.319
L2 = 14.375
L3 = 16317.5
L4 = 0.986772

```

Результати роботи консольної програми:

Схема алгоритму:



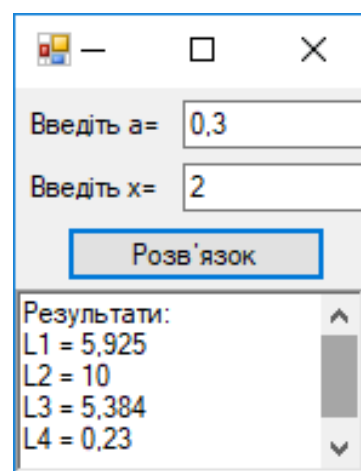
Текст програми для проекту з формою та його результати:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    richTextBox1->Text="Результати:\n";
    double x, a, L;    int k = 1;
    a = Convert::ToDouble(textBox1->Text);
    x = Convert::ToDouble(textBox2->Text);
M1: switch (k)
    { case 1: L = Math::Pow(x, 2+a) + 1; break;
      case 2: L = (x+1) / a; break;
      case 3: L = Math::Exp(x-a) - Math::Pow(a, x); break;
      case 4: L = Math::Log10(Math::Abs(x-a)); break;
    }
    richTextBox1->Text += "L"+k.ToString()+" = "+L.ToString("0.###")+"\n";
    k++;
    if (k<=4) goto M1;
}
```

Використання оператора

```
if (k<=4) goto M1;
```

у цій програмі організовує чотириразове повторення виконання групи операторів, починаючи від мітки M1, що фактично утворює псевдоцикл.



Приклад 4.27. Ввести значення x та обчислити

$$y = \begin{cases} \sin(e^{|a+c|}) + x^{2+b} & \text{за } x < a; \\ \sqrt[3]{a + |5.3b|} + c & \text{за } a \leq x < b; \\ \cos^2(a+b) + \sin x^2 & \text{за } x > b \end{cases}$$

для одного з трьох варіантів параметрів:

- 1) $a = -3,7$; $b = 5,6$; $c = e^{\cos x}$;
- 2) $a = 0,81$; $b = -2,4$; $c = \operatorname{tg}|bx|$;
- 3) $a = 2,5$; $b = 0,6$; $c = \operatorname{ctg}|bx|$.

Текст програми, схема алгоритму та скріншоти результатів роботи програми в консолі Visual C++:

```
#include <iostream>
#include <math.h>
#include <conio.h>
using namespace std;
int main()
{ setlocale(0, ".1251");
  system("color F0");
  double x, y, a, b, c; int k, n;
  cout<<" Введіть значення x = ";
  cin>> x;
  cout<<" Введіть номер варіанта даних (1, 2 або 3) = ";
  cin>>n;
```

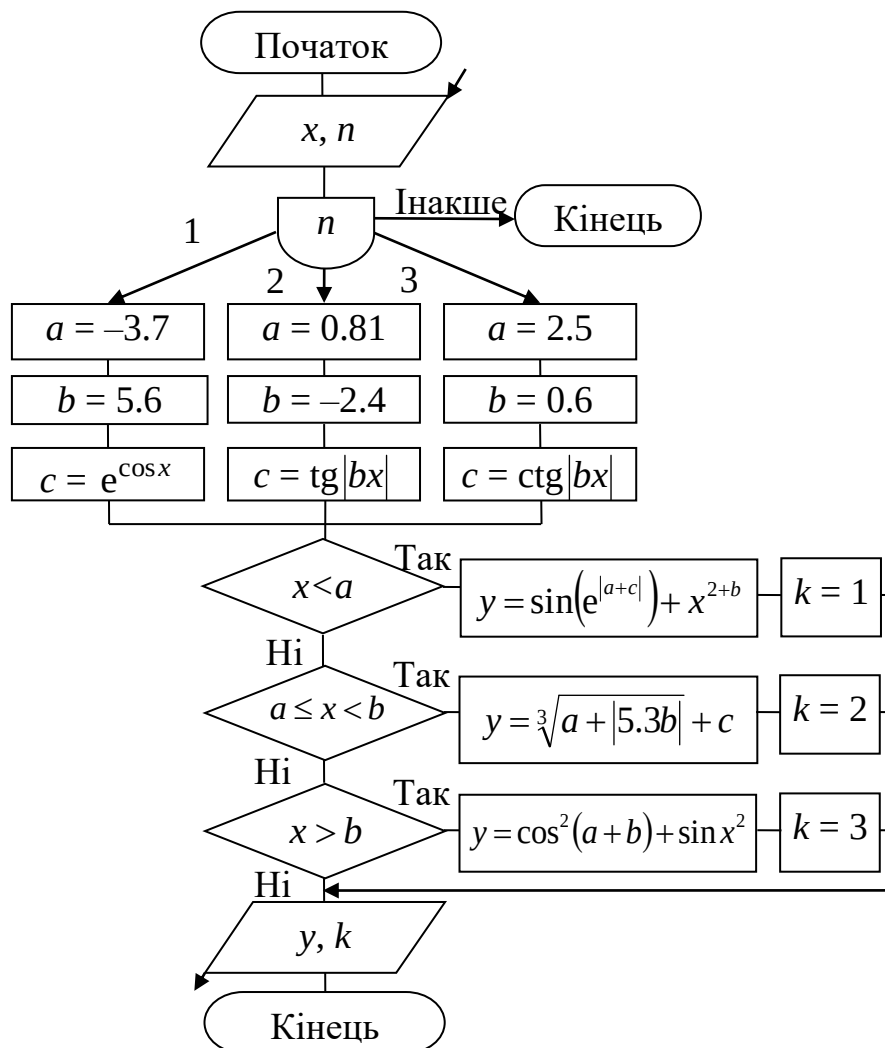
```
Введіть значення x = 0.34
Введіть номер варіанта даних (1, 2 або 3) = 3
Результати: 3-й варіант: y = -0,868 умова №1
->
Для завершення натисніть <Enter>
```

```

cout<<" Результати:";
switch (n)
{ case 1: a=-3.7; b=5.6; c=exp(cos(x)); break;
  case 2: a=0.81; b=-2.4; c=tan(fabs(b*x)); break;
  case 3: a=2.5; b=0.6; c=1/tan(fabs(b*x)); break;
  default: puts(" Було введено некоректний номер"); _getch(); return 0;
}
if (x<a) { y=sin(exp(fabs(a+c)))+pow(x,2+b); k=1;}
if (x>=a &&
x<b) {
y=pow(a+fabs(5.
3*b),1./3)+c;
k=2;}
if (x>=b) { y=pow(cos(a+b),2)+sin(x*x); k=3;}
printf(" %i-й варіант: y = %5.3f умова №%i\n",n,y,k);
puts(" ->\n Для завершення натисніть <Enter> \n");
_getch();
return 0;
}

```

Введіть значення $x = 1$
Введіть номер варіанта даних (1, 2 або 3) = 6
Результати: Було введено некоректний номер



Приклад 4.28. Визначити за введеною датою день тижня, на який ця дата припадає. Для обчислення користуватися формулою

$$\left(d + \left\lfloor \frac{13m-1}{5} \right\rfloor + y + \left\lfloor \frac{y}{4} \right\rfloor + \left\lfloor \frac{c}{4} \right\rfloor - 2c + 777 \right) \bmod 7,$$

де d – день місяця;

m – номер місяця, починаючи рахунок від березня, як це робили у Стародавньому Римі (березень – 1, квітень – 2, ..., лютий – 12);

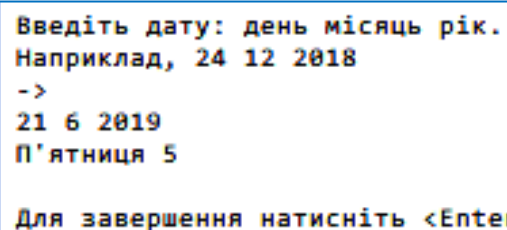
y – рік у столітті;

c – кількість століть.

Квадратні дужки у формулі означають, що слід взяти лише цілу частину від значення у дужках. Обчислене за формулою значення визначає день тижня: 1 – понеділок, 2 – вівторок, ..., 6 – субота, 0 – неділя.

Текст програми для консольної програми:

```
#include <iostream>
#include <conio.h>
using namespace std;
int main()
{ setlocale(0, ".1251");
  system("color F0"); // Встановити білий фон і чорний текст
  int day, month, year; // день, місяць, рік
  int c, y;             // сторіччя і рік у сторіччі
  int m, d;             // місяць і день тижня за римським календарем
  puts(" Введіть дату: день місяць рік.");
  puts(" Наприклад, 24 12 2018 "); puts(" ->");
  scanf("%i %i %i", &day, &month, &year);
  if (month==1||month==2) year--; //січень і лютий належать до минулого року
  m = month-2;                   //рік розпочинається з березня
  if (m<=0) m+=12;               //для січня і лютого
  c = year/100;
  y = year - c*100;
  d = (day+(13*m-1)/5+y+y/4+c/4-2*c+777)%7;
  switch (d)
  { case 1: puts(" Понеділок 1 "); break;
    case 2: puts(" Вівторок 2 "); break;
    case 3: puts(" Середа 3 "); break;
    case 4: puts(" Четвер 4 "); break;
    case 5: puts(" П'ятниця 5 "); break;
    case 6: puts(" Субота 6 "); break;
    case 0: puts(" Неділя 7 ");
  }
  printf("\n Для завершення\n        натисніть <Enter> \n");
  _getch();
  return 0;
}
```



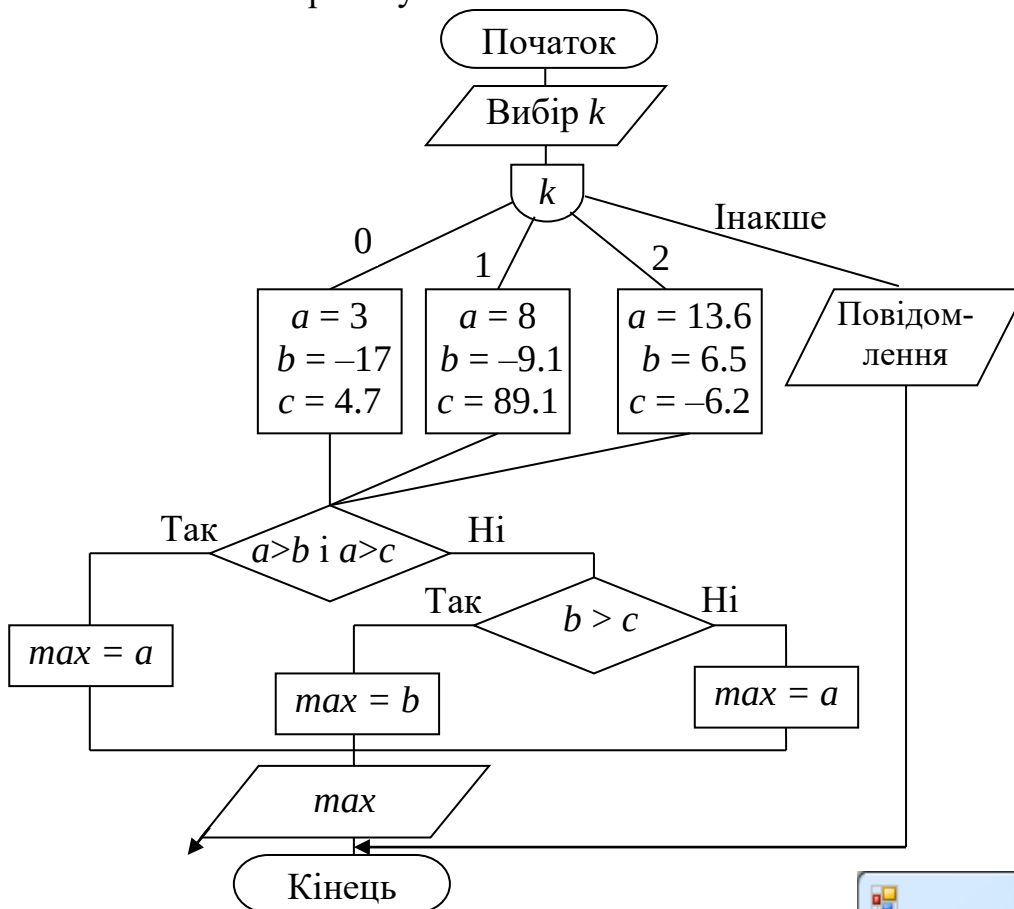
```
Введіть дату: день місяць рік.
Наприклад, 24 12 2018
->
21 6 2019
П'ятниця 5
Для завершення натисніть <Enter>
```

Приклад 4.29. Для будь-якого варіанта з трьох чисел визначити і вивести найбільше з них:

- 1) $a = 3$; $b = 17$; $c = -4.7$;
- 2) $a = 8$; $b = -9.1$; $c = 89.1$;
- 3) $a = 13.6$; $b = 6.5$; $c = -6.2$.

Розв'язок. Пошук найбільшого числа зорганізуємо за допомогою умовної операції $?:$, а вибір варіантів чисел – за допомогою елемента-списка `listBox`, для якого у властивості `Items` сформуємо три рядки з наборами даних. Далі в програмному коді буде опитуватись значення номера вибраного елемента списку (властивість `SelectedIndex`), нумерація яких розпочинається з 0.

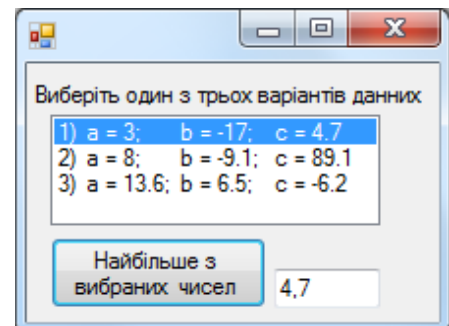
Схема алгоритму:



Текст програми:

```

private System::Void button1_Click(System::Object^
    sender, System::EventArgs^ e)
{
    double a, b, c, max;
    int k = listBox1->SelectedIndex;
    switch(k)
    {
        case 0: a=3; b=-17; c=4.7; break;
        case 1: a=8; b=-9.1; c=89.1; break;
        case 2: a=13.6; b=6.5; c=-6.2; break;
        default: MessageBox::Show("Виберіть один з трьох варіантів даних");
                return;
    }
}
  
```



```
max = (a>b && a>c)? a : (b>c)? b : c;  
textBox1->Text = Convert::ToString(max);  
}
```

Конструкцію з двома вкладеними одна в одну умовними операціями ? :

```
max = (a>b && a>c)? a : (b>c)? b : c;
```

можна замінити двома вкладеними один в один операторами **if-else**, але така конструкція є більш громіздкою:

```
if(a>b && a>c) max=a;  
else if(b>c) max=b;  
else max=c;
```

4.3 Програмування циклічних алгоритмів

4.3.1 Циклічні алгоритми

Обчислювальний процес називається *циклічним*, якщо він неодноразово повторюється, допоки виконується певна задана умова. Блок повторюваних операторів називають *тілом* циклу. Існують три різновиди операторів циклу:

- ✓ оператор циклу **for**;
- ✓ оператор циклу з передумовою **while**;
- ✓ оператор циклу з післяумовою **do-while**.

Всі оператори циклу неодмінно містять такі складові частини:

- ✓ присвоювання початкових значень (ініціалізація);
- ✓ умова продовження циклу;
- ✓ тіло циклу;
- ✓ зміна параметра циклу.

4.3.2 Оператор циклу з параметром for

Цей оператор зазвичай використовується, коли є заздалегідь відома кількість повторювань, або коли умова продовження виконання циклу записується коротким виразом. За приклади використання даного оператора можуть слугувати обчислення сум заданої кількості доданків, пошук мінімального (максимального) елемента послідовності чисел, зчитування рядків **richTextBox**, сортування елементів масиву за зростанням (за спаданням) значень тощо.

Синтаксис оператора:

```
for (<ініціалізація>; <умова>; <модифікації>) <тіло циклу>;
```

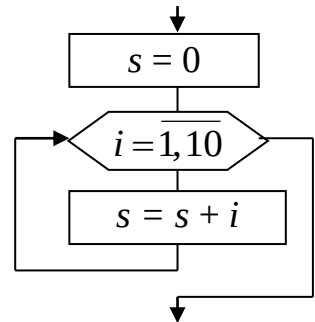
Конструктивно цей оператор складається з трьох основних блоків, розміщених у круглих дужках і відокремлених один від одного крапкою з комою (;), та команд (тіла циклу), які мають багаторазово виконуватись у цьому циклі. На початку виконання оператора циклу одноразово у блоці *ініціалізації* задаються початкові значення змінних (параметрів), які керують циклом. Потім перевіряється *умова* і, якщо вона виконується (**true** чи має ненульове значення), то виконується *команда* (чи група команд в операторних дужках {}) тіла циклу. *Модифікації* змінюють параметри циклу і, в разі істинності умови, виконання

циклу триває. Якщо умова не виконується (**false** чи дорівнює нулю), відбувається вихід із циклу і керування передається на оператор, який слідує за оператором **for**. Суттєвим є те, що перевірка умови виконується на початку циклу. Це означає, що тіло циклу може не виконуватись жодного разу, якщо умова спочатку є хибна. Кожне повторення (крок) циклу називається *ітерацією*.

Простий приклад для обчислення суми $S = \sum_{i=1}^{10} i$ проілюструє використання оператора **for**:

```
int s = 0;
for (int i = 1; i <= 10; i++) s += i;
```

Цей оператор можна прочитати так: «виконати команду $s += i$ 10 разів (для значень i від 1 до 10 включно, де i на кожній ітерації збільшується на 1)». У наведеному прикладі є два присвоювання початкових значень: $s=0$ і $i=1$, умова продовження циклу: $(i \leq 10)$ і змінення параметра: $i++$. Тілом циклу є команда $s += i$.



Порядок виконання цього циклу комп'ютером є такий:

- 1) присвоюються початкові значення ($s=0$, $i=1$);
- 2) перевіряється умова ($i \leq 10$);
- 3) якщо умова є істинна (**true**), виконується команда (чи команди) тіла циклу: до суми, обчисленої на попередній ітерації, додається нове число;
- 4) параметр циклу збільшується на 1.

Далі повертаємось до пункту 2. Якщо умову у пункті 2 не буде виконано (**false**), станеться вихід із циклу.

В операторі можливі конструкції, коли є відсутній той чи інший блок: *ініціалізація* може бути відсутня, якщо початкове значення задати попередньо; *умова* – якщо припускається, що умова є завжди істинна, тобто слід неодмінно виконувати тіло циклу, допоки не зустрінеться оператор **break**; а *модифікації* – якщо приріст параметра здійснювати в тілі циклу чи взагалі це є непотрібне. Тоді сам вираз блока пропускається, але крапка з комою (;) неодмінно має залишитись. Можливою є наявність *порожнього* оператора (оператор є відсутній) у тілі циклу. Наприклад, суму з попереднього прикладу можна обчислити в інший спосіб:

```
for (int s = 0, i = 1; i <= 10; s += i++);
```

У цьому прикладі є відсутній *оператор*, а блок *ініціалізації* вмістить два оператори, розділених операцією «кома», які задають початкові значення змінних s та i .

Розглянемо приклади циклів **for**.

- 1) Виведення чисел від 1 до 10 з їхніми квадратами:

```
for (int i=1; i<11; i++) cout<< i << " " << i*i << "\n";
```

Ця сама дія з виведенням до richTextBox матиме вигляд:

```
for (int i=1; i<11; i++)
  richTextBox1->Text += i.ToString()+" "+(i*i).ToString()+"\n";
```

1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

Цикл виконуватиметься за $i = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$. Після цього i набуде значення 11, умова $(11 < 11)$ не виконується (**false**) і відбувається вихід з циклу.

2) Виведення додатних непарних цілих від 1 до 100 з їхніми квадратами. Цикл записується аналогічно до попереднього, але параметр збільшуватиметься на 2, тобто цикл виконуватиметься для значень $i = 1, 3, 5, 7, \dots, 97, 99$.

```
for (int i=1; i<100; i+=2)
    cout<< i << " " << i*i << "\n";
```

3) Обчислення факторіала $F=n!$ (нагадаємо, що факторіал обчислюється за формулою $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-2) \cdot (n-1) \cdot n$, наприклад: $4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$). Наведемо три аналогічні за дією форми оператора **for**:

a) `int F=1, n=5; for(int i=1; i<=n; i++) F *= i;`
 б) `int F, i, n=5; for(F=1, i=1; i<=n; F *= i++);`
 в) `int F=1, i=1, n=5; for( ; i<=n; ) F *= i++;`

Для завчасного початку чергової ітерації циклу можна використати оператор переходу до наступної ітерації **continue** (див. п. 4.3.5), наприклад:

```
for (i = 0; i < 20; i++)
{ if (a[i] == 0) continue;
  a[i] = 1/a[i];
}
```

Для завчасного виходу з циклу застосовують оператори **break** (вихід з конструкції) (див. п. 4.3.5), **goto** (безумовний перехід) (див. п. 4.2.2) і **return** (вихід з поточної функції) (див. підрозд. 8.1).

Деякі варіанти застосування оператора **for** підвищують його гнучкість за рахунок можливості використання декількох змінних-параметрів циклу.

Наприклад:

```
int main ( )
{ int top, bot;   char string[100], temp;
  for ( top=0, bot=98 ; top<bot ; top++, bot--)
  { temp = string[top];
    string[top] = string[bot];
    string[bot] = temp;
  }
  return 0;
}
```

У цьому прикладі для реалізації записування рядка символів у зворотному порядку параметрами циклу є дві змінні `top` та `bot`, значення яких рухаються назустріч один одному. Зауважимо, що у блоці ініціалізації оператора **for** задаються через кому початкові значення відразу обох змінних (параметрів). Так само через кому записано модифікацію цих змінних після умовного виразу.

Наведемо ще один варіант оператору циклу **for**, в якому задається початкове значення параметра `i`, його модифікація, проте в умовному виразі ця змінна фігурує лише як індекс елемента масиву:

```
for (i=0; t[i]>=0; i++) ;
```

У цьому операторі параметр циклу `i` набуде значення номера першого у масиві від'ємного елемента масиву `t`. Отже, такий цикл може використовуватись, наприклад, для пошуку елементів за певним значенням.

Іншим цікавим варіантом застосування оператора `for` є нескінченний цикл. Для організації такого циклу можна записати пустий умовний вираз, а для виходу з циклу скористатись умовним оператором `if` разом з оператором `break`.

Наприклад:

```
for ( ; ; )
{
    . . .
    if ( <деяка умова> ) break;
    . . .
}
```

Розглянемо кілька прикладів розв'язування задач, в яких є доцільне використання оператора циклу `for`. Для усіх програм розроблено алгоритмічні схеми (блок-схеми), які визначають порядок виконання дій, і наведено форми з результатами обчислень.

Приклади проектів програм з оператором циклу `for` в C++

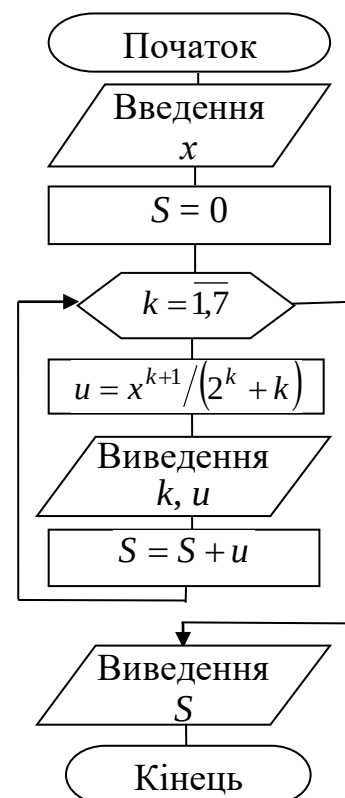
Приклад 4.30. Обчислити суму $s = \sum_{k=1}^7 \frac{x^{k+1}}{2^k + k}$,

значення x ввести з екрана.

Розв'язок. Щоби впевнитися, що сума обчислюватиметься з усіх семи доданків, спочатку виведемо кожен з них, після чого виведемо вже саму суму.

Текст програми в онлайн-компіляторі та блок-схема:

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
    double u, s = 0, x;
    cout << "Input x= ";
    cin >> x;
    for (int k = 1; k <= 7; k++)
    {
        u = pow(x, k+1)/(pow(2, k)+k);
        cout << k << ": " << u << endl;
        s = s + u;
    }
    cout << "Result s = " << s ;
    return 0;
}
```



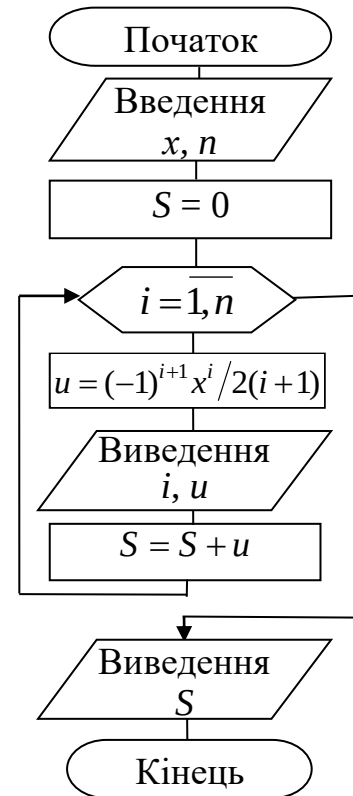
```
Input x= 2.92
1:  2.84213
2:  4.14951
3:  6.60905
4:  10.6141
5:  16.7531
6:  25.8572
7:  39.1498
Result s = 105.975
```

Приклад 4.31. Ввести ціле число n і дійсне число x , обчислити суму ряду

$$s = \frac{x}{4} - \frac{x^3}{6} + \frac{x^5}{8} - \dots = \sum_{i=1}^n \frac{(-1)^{i+1} x^{2i-1}}{2(i+1)}.$$

Текст консольної програми у Visual C++ та блок-схема:

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
    setlocale(0, ".1251");
    double x, u, s=0;
    int i, n;
    cout<<"Введіть ціле значення n= "; cin>> n;
    cout<<"Введіть значення x= "; cin>> x;
    cout<< "\nРезультати:\n";
    for(i=1; i<=n; i++)
    { u = pow(-1,i+1.0)*pow(x,i)/(2*(i+1));
      cout<< "Доданок " << i << " = " << u << endl;
      s+=u;
    }
    cout<< "Сума = " << s << endl;
    system("pause>>void");
    return 0;
}
```



Результат виконання програми:

```
Введіть ціле значення n= 5
Введіть значення x= 2.4

Результати:
Доданок 1 = 0.6
Доданок 2 = -0.96
Доданок 3 = 1.728
Доданок 4 = -3.31776
Доданок 5 = 6.63552
Сума = 4.68576
```

Приклад 4.32. Обчислити добуток $z = \prod_{k=-2}^4 \frac{(k+1)(k-5)}{k(k+6)}$. При виконуванні

обчислень не слід враховувати ті множники, в яких чисельник чи знаменник дорівнюють нулю.

Розв'язок. При обчисленні добутку змінна k по чергові набуватиме таких значень: $-2, -1, \dots, 3, 4$. За значення $k = -1$ утвориться нуль у чисельнику, а отже, весь добуток перетвориться на нуль, а за $k = 0$ – утвориться нуль у знаменнику. Тому, щоб вилучити ці множники, слід застосувати умовний оператор `if`, який можна організувати у два способи:

```
if (k != -1 && k != 0)
    z *= (double)(k+1)*(k-5)/(k*(k+6));
```

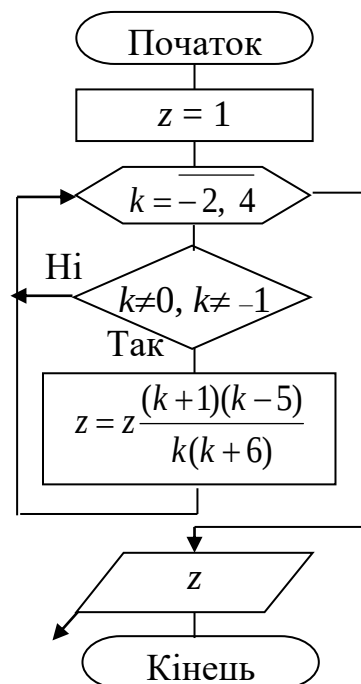
або

```
if (k==0 || k==-1) continue;
else z *= (double)(k+1)*(k-5)/(k*(k+6));
```

Умову `if(k!=0)` можна записати простіше, `if(k)`, оскільки ненульове значення розцінюється як `true`.

Текст програми в онлайн-компіляторі та блок-схема:

```
#include <iostream>
using namespace std;
int main()
{
    double z = 1;
    for (int k = -2; k <= 4; k++)
        if (k != -1 && k != 0)
            z *= (double)(k+1)*(k-5)/(k*(k+6));
    cout<< "Result z = "<< z ;
    return 0;
}
```



Результат виконання програми:

Result z = -0.0208333

Приклад 4.33. Написати програму, яка обчислює часткову суму ряду: $1 - 1/3 + 1/5 - 1/7 + 1/9 - \dots$ для заданої кількості членів ряду, яка вводиться користувачем (при підсумовуванні достатньо великої кількості членів цього ряду, значення часткової суми наближується до $\pi/4$).

Розв'язок. Для обчислення запишемо формулу для обчислення доданка ряду $\pm \frac{1}{2i-1}$, де $i = 1, 2, 3, \dots$

Текст консольної програми у Visual C++:

```
#include <iostream>
using namespace std;
int main()
{
    setlocale(0, ".1251");
    system("color F0");
    // Обчислення числа π з використанням властивості ряду 1 - 1/3 + 1/5 - 1/7 + ...
    double x; int n; // Член ряду та кількість доданків у сумі
    float sum=0; // Часткова сума, значення якої при підсумовуванні
    // вельми значної (понад 1000) кількості елементів ряду, наближається до π/4
    printf("Обчислення суми ряду 1 - 1/3 + 1/5 - 1/7 + ... \n");
    printf("Введіть кількість членів ряду для суми -> ");
    scanf("%i", &n);
}
```

```

for (int i = 1; i <= n; i++)
{ x = 1./(2*i - 1) ;
  if ((i % 2)==0) x = -x;
  sum += x;
}
printf("Сума ряду: %2.6f\n", sum);
printf("Обчислене значення числа ПІ = %2.6f\n", sum * 4);
printf("\nДля завершення натисніть [Enter]");
_getch();
return 0;
}

```

Результат виконання програми:

Обчислення суми ряду $1 - 1/3 + 1/5 - 1/7 + \dots$
 Введіть кількість членів ряду для суми -> 10000
 Сума ряду: 0.785375
 Обчислене значення числа ПІ = 3.141499
 Для завершення натисніть [Enter]

Приклад 4.34. Обчислити суму всіх двоцифрових чисел, кратних до числа 3.

Розв'язок. Двоцифрові числа – це цілі числа від 10 до 99 включно. Змінюючи параметр циклу від 12 (перше двоцифрове число, кратне до 3) до 99 (останнє двоцифрове число, кратне до 3) із кроком 3, можна використовувати його як число для додавання до суми.

Текст програми в онлайн-компіляторі:

```

#include <iostream>
using namespace std;
int main()
{
  int i, s=0;           // Спочатку сума дорівнює 0
  for (i=12;i<=99;i+=3) s+=i; // Обчислити суму чисел від 12 до 99 з кроком 3
  cout<<"Сума двоцифрових чисел, кратних до числа 3, дорівнює "<<s<<endl;
  return 0;
}

```

Результат виконання програми:

Сума двоцифрових чисел, кратних до числа 3, дорівнює 1665

Приклад 4.35. Вивести таблицю степенів двійки від нульової до десятої.

Розв'язок. Для обчислення і виведення усіх значень степенів двійки достатньо організувати лише один цикл.

Текст програми:

```

#include <iostream>
using namespace std;
int main()
{ int n; // Показник степеня

```

```

int st; // Значення 2 у степені n
printf("Таблиця степенів двійки \n");
for (n=0, st=1; n<=10; n++)
{ printf("%3i %5i\n", n, st);
  st*=2;
}
printf("Для завершення натисніть [Enter]\n");
return 0;
}

```

Результат виконання програми:

Таблиця степенів двійки

0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Для завершення натисніть [Enter]

Приклад 4.36. Дослідження функцій на певному проміжку (табулювання).

Скласти схему алгоритму і програму табулювання функцій (дослідження на певному проміжку) $y(x) = \frac{x^3 \cdot \cos^2 x}{2^x}$ та $z(x) = 2 \sin^3(2x) \ln(0.5 + x)$, змінюючи x на проміжку $[1, 11]$ з кроком $h = 0,2$.

Розв'язок. Доволі часто при дослідженні функціональних залежностей виникає потреба побудови графіка $y = f(x)$ чи обчислення значень $y = f(x)$ на заданому проміжку $x \in [A, B]$ з кроком h .

Для графічного відображення графіків обох функцій доцільно створити програмний проект з формою, на який розмістити відповідні елементи:

- три елементи `textBox` для введення початкового (змінна A), кінцевого (змінна B) значень x та кроку (змінна h), з яким буде змінюватись x . Для створення відповідного пояснювального підпису біля кожного `textBox` розмістити три елементи `label`;

- командну кнопку `button`, при натисненні на яку буде виконуватись програмний код табулювання функцій;

- елемент `richTextBox` для виведення результатів у вигляді трьох стовпців. Установити для його властивості `Scrollbars` (прокрутки) значення `Both`;

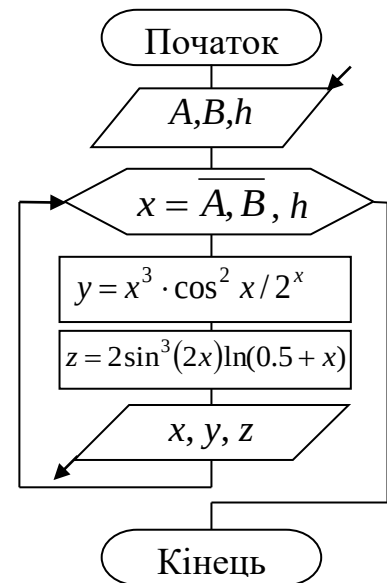
- елемент `chart` (з групи *Данные*) для виведення діаграми. Налаштувати його, створивши на ньому два графіки (`Series`) з типом `Spline`. Для цього клацнути у властивості `Series` (Коллекция) , що призведе до відкриття діалого-

вого вікна *Редактор колекції Series*, в якому для першого графіка (Series1) вибрати у властивості ChartType тип графіка Spline (замість значення Column). У цьому саме вікні знайти ще одну властивість – LegendText, щоб записати в ній назву першого графіка: $y = f(x)$. Також можна змінити колір (властивість Color), тип лінії (властивість BorderDashStyle) та інші параметри графіка.

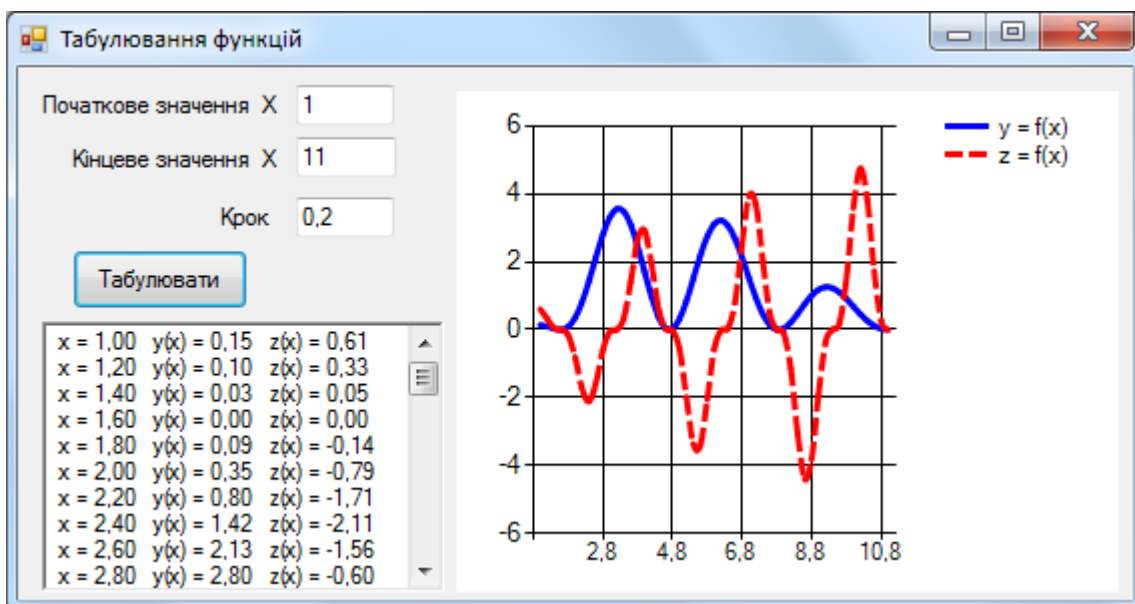
Після цього натиснути кнопку *Добавить*, щоб створити графік Series2, та задати для нього такий самий тип і назву другого графіка – $z = f(x)$. Наприкінці натиснути кнопку *OK*, щоб закрити вікно *Редактор колекції Series*.

Текст програми та блок-схема:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    richTextBox1->Clear();
    chart1->Series[0]->Points->Clear();
    chart1->Series[1]->Points->Clear();
    double x, y, z, A, B, h;
    A=Convert::ToDouble(textBox1->Text);
    B=Convert::ToDouble(textBox2->Text);
    h=Convert::ToDouble(textBox3->Text);
    for(x=A; x <=B+0.1*h; x+=h)
    {
        y=x*x*x*Math::Pow(Math::Cos(x),2)/Math::Pow(2,x);
        z=2*Math::Pow(Math::Sin(2*x),3)*Math::Log(0.5+x);
        chart1->Series[0]->Points->AddXY(x, y);
        chart1->Series[1]->Points->AddXY(x, z);
        richTextBox1->Text += "x = " + x.ToString("0.00")
            + " \t y(x) = " + y.ToString("0.00")
            + " \t z(x) = " + z.ToString("0.00") + Environment::NewLine;
    }
}
```



Результати роботи:



Приклад 4.37. Написати програму обчислення напруги ланцюга $R(t)$, яка змінюється упродовж певного часу t за законом:

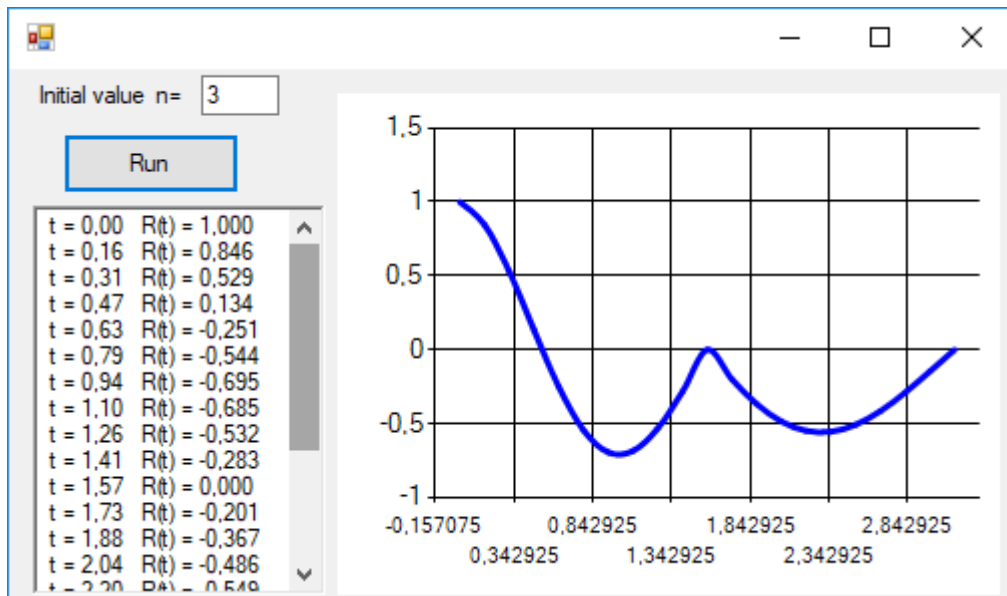
$$R(t) = \begin{cases} e^{(-t/n)} \cos(nt) & \text{за } 0 \leq t < \frac{\pi}{2}; \\ e^{(-t/(n+1))} \sin((n-1) \cdot t) & \text{за } \frac{\pi}{2} \leq t \leq \pi. \end{cases}$$

якщо t змінюється на проміжку $[0, \pi]$ з кроком $h = \pi/20$. Параметр n ввести з клавіатури.

Текст програми:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    richTextBox1->Clear();
    chart1->Series[0]->Points->Clear();
    const double Pi=3.1415;
    double t, R, h=Pi/20;
    int n=Convert::ToInt32(textBox1->Text);
    for(t=0; t<=Pi+0.01*h; t+=h)
    {
        if (t>=0 && t<Pi/2) R = Math::Exp(-t/n)*Math::Cos(n*t);
        else
            if (t<=Pi) R = Math::Exp(-t/(n+1))*Math::Sin((n-1)*t);
        richTextBox1->Text += "  t = " + t.ToString("0.00") +
            "    R(t) = " + R.ToString("0.000") + Environment::NewLine;
        chart1->Series[0]->Points->AddXY(t, R);
    }
}
```

Результати роботи:



Приклад 4.38. Написати програму наближеного обчислення інтегралу функції $f(x) = x^3 - x^2 - 4.4x + 4$ трьома методами: прямокутників, трапецій та Сімпсона на заданому інтервалі $x \in [a, b]$ із заданою точністю ε .

Розв'язок. У багатьох наукових та інженерних задачах треба віднайти значення інтегралу функції $f(x)$, де x змінюється на проміжку $[a, b]$. Якщо для неперервної на проміжку підінтегральної функції можна віднайти первісну функцію $F(x)$, то визначений інтеграл легко обчислити за формулою Ньютона-Лейбніца:

$$\int_a^b f(x)dx = F(b) - F(a)$$

Але у багатьох випадках не вдається знайти первісну функцію або вона є настільки складною, що обчислювати інтеграл за її допомогою важче, ніж у інші способи. У таких випадках застосовують методи чисельного інтегрування, які базуються на тому, що значення інтегралу дорівнює площі криволінійної трапеції, розміщеної поміж лінією графіка функції та віссю абсцис (рис. 4.1). Згідно з цими методами, проміжок інтегрування $[a, b]$ слід поділити на кілька відрізків (наприклад, n відрізків), та замінити площу S зазначеної криволінійної трапеції на суму площин маленьких криволінійних трапецій s_i , на яких задану функцію $f(x)$ замінено наближеною функцією $f_i(x)$, тобто

$$S = \int_a^b f(x)dx = \sum_{i=0}^{n-1} s_i.$$

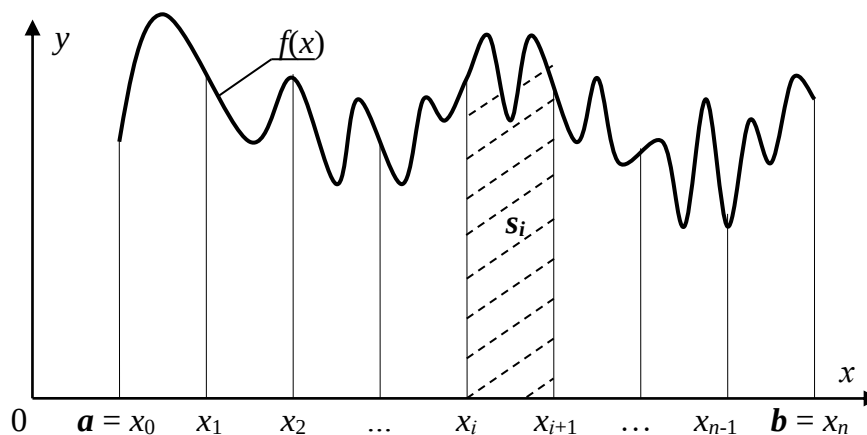


Рис. 4.1. Криволінійні трапеції для інтегрування

Залежно від того, якими є наближені функції $f_i(x)$, існують різні методи (формули) обчислення визначених інтегралів. В усіх цих методах інтеграл наближено обчислюють як певну сукупність значень підінтегральної функції $f(x_i)$, де $x_i = a + ih$ – межі відрізків інтегрування, $h = x_{i+1} - x_i$, – крок інтегрування, $i = 0, 1, \dots, n$. Найбільш відомими в інженерній практиці є методи чисельного інтегрування: прямокутників, трапецій та Сімпсона.

Для чисельного обчислення інтегралів *методом прямокутників* задану функцію $f(x)$ на кожному відрізку $h = x_{i+1} - x_i$ (де $i = 0, 1, \dots, n$) замінюють на прямі лінії, паралельні до осі абсцис (рис. 4.2). При цьому криволінійна трапеція замінюється на n прямокутників.

$$S = \sum_{i=0}^{n-1} s_i = \sum_{i=0}^{n-1} (x_{i+1} - x_i) f(x_i) = h \sum_{i=0}^{n-1} f(x_i).$$

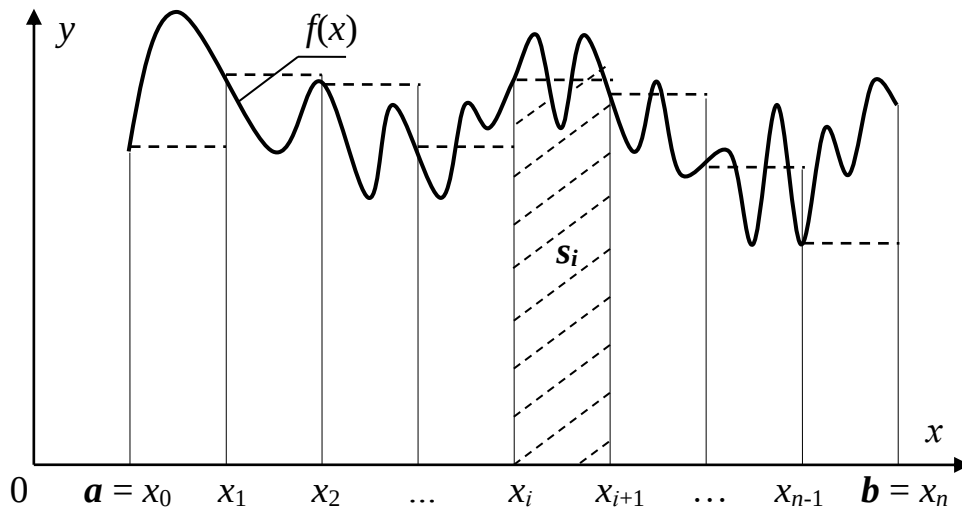


Рис. 4.2. Інтерпретація інтегрування методом прямокутників

В основу *формули трапецій* покладено заміну кривої підінтегральної функції на ламану, тобто з'єднуються прямими лініями значення функцій на кінцях відрізків. Отже, площу криволінійної трапеції наближено замінюємо на суму площин n трапецій (рис. 4.3).

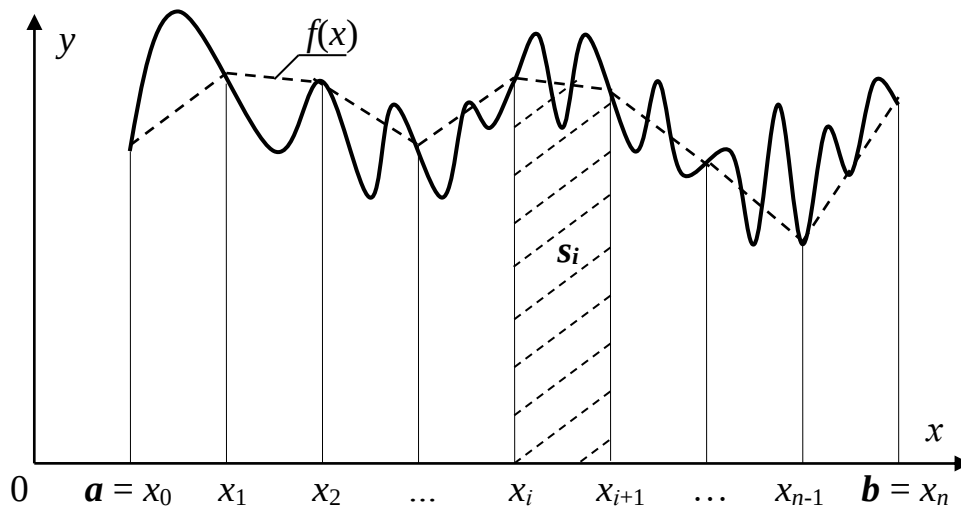


Рис. 4.3. Інтерпретація інтегрування методом трапецій

Загальна площа трапецій S і відповідно наближене значення інтегралу за методом трапецій обчислюється за формулою

$$S = \sum_{i=0}^{n-1} s_i = \sum_{i=0}^{n-1} \frac{h}{2} (f(x_i) + f(x_{i+1})) = \frac{h}{2} \left(f(a) + f(b) + 2 \sum_{i=1}^{n-1} f(x_i) \right).$$

Для наближеного обчислення інтеграла за *методом Сімпсона* крива підінтегральної функції замінюється на відрізки квадратичних парабол, проведених через кінці кожних трьох сусідніх ординат значень функції, при цьому весь

проміжок інтегрування розбивають на парну кількість n відрізків $[x_i - h, x_i + h]$. Отже, площу криволінійної трапеції наближено замінюємо на суму $n/2$ площин під параболami (рис. 4.4).

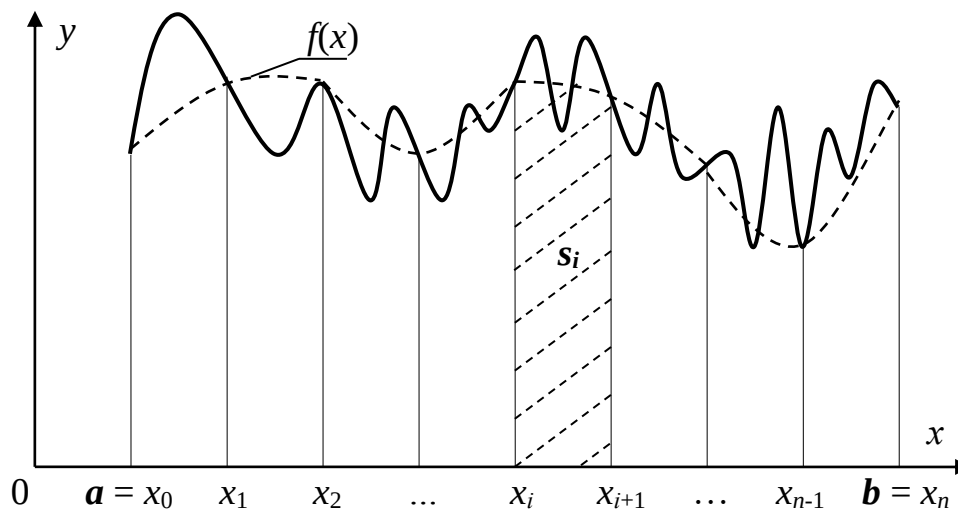


Рис. 4.4. Інтерпретація інтегрування методом Сімпсона

Остаточна формула обчислення інтегралу S методом Сімпсона:

$$S = \frac{h}{3} \left(f(a) + f(b) + 4 \sum_{i=1}^{n-1} f(x_i) + 2 \sum_{j=1}^{n-2} f(x_j) \right),$$

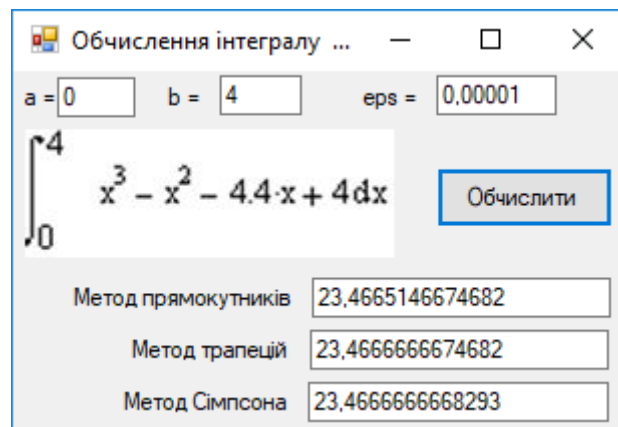
де i – непарні числа, j – парні числа.

Порівнюючи похибки наведених трьох методів, зазначимо, що метод Сімпсона дає точний результат для поліномів до третього ступеня включно, а методи прямокутників та трапецій є точними лише за інтегрування поліномів ступеня не вище за перший.

Зображення інтегралу на формі організовано за допомогою елемента керування PictureBox. Зображення у форматі *jpg* завантажено до властивості Image цього елемента.

Текст програми:

```
// Підінтегральна функція
double f(double x)
{ return x*x*x-x*x-4.4*x+4;
}
// Метод прямокутників
double integral_Pr(double a, double b, double eps)
{ double h=eps,s,x;
  for (s=0, x=a; x<b ; x+=h) s+=f(x);
  return h*s;
}
```



```

// Метод трапецій
double integral_Trap(double a,double b,double eps)
{ double h=eps,s,x;
  for (s=0, x=a+h; x<b ; x+=h) s+=f(x);
  return h/2*(f(a)+f(b)+2*s);
}

// Метод Сімпсона
double integral_Simp(double a,double b,double eps)
{ double h=eps,s1,s2,x; int i, n;
  n=(int)((b-a)/h)+1;
  if (n%2==0) {n++;h=(b-a)/(n-1);}
  for (s1=0, i=1,x=a+h; i<n-1 ; i+=2,x+=2*h) s1+=f(x);
  for (s2=0, i=2,x=a+2*h; i<n-2 ; i+=2, x+=2*h) s2+=f(x);
  return h/3*(f(a)+f(b)+4*s1+2*s2);
}

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
  double t, R, Pi=3/1415, h=Pi/20;
  double a,b,i_bar,i_trap,i_simp,eps;
  a=Convert::ToDouble(textBox1->Text);
  b=Convert::ToDouble(textBox2->Text);
  eps=Convert::ToDouble(textBox3->Text);
  i_bar= integral_Pr(a,b,eps); // Інтеграл методом прямокутників
  textBox4->Text=i_bar.ToString();
  i_trap= integral_Trap(a,b,eps); // Інтеграл методом трапецій
  textBox5->Text=i_trap.ToString();
  i_simp= integral_Simp(a,b,eps); // Інтеграл методом Сімпсона
  textBox6->Text=i_simp.ToString();
}

```

Приклад 4.39. У програмі організовано введення 12 цілих чисел та обчислюється добуток парних з цих чисел.

Розв'язок. Існує коло задач, в яких необхідно певним чином опрацювати задану числову послідовність, причому для обчислення результату достатньо переглянути послідовність один раз. Наприклад, щоб обчислити середнє арифметичне заданої послідовності чисел, можна підсумовування чисел і підрахунок їхньої кількості поєднати з введенням чисел. Тоді не потрібно буде зберігати всю послідовність в пам'яті комп'ютера (у вигляді масиву), достатньо мати одну скалярну змінну цілого або дійсного типу і по черзі присвоювати їй значення, які вводяться.

Числова послідовність може задаватися зі зазначенням кількості чисел або мати якусь ознаку кінця. Цей та наступних декілька прикладів присвячені програмному опрацюванню послідовностей введених чисел.

У цій програмі у циклі, який повторюється 12 разів, виконуються такі дії:

- ✓ виводиться запрошення для введення наступного числа;
- ✓ вводиться нове число;
- ✓ перевіряється введене число на парність і, якщо це так, то число пере-

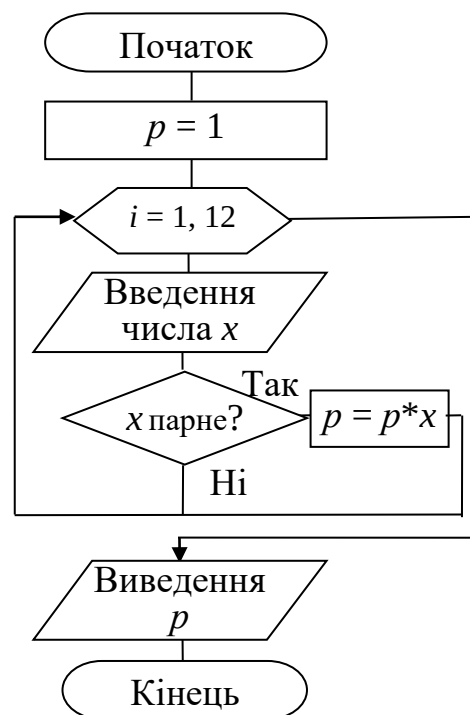
множується на добуток.

Текст консольної програми у Visual C++ та блок-схема:

```
#include <iostream>
using namespace std;
int main()
{
    setlocale(0, ".1251");
    int i, x, p=1;                                     // Спочатку добуток дорівнює 1.
    for (i=1; i<=12; i++)                             // У циклі 12 разів повторити дії:
    { cout << "Ввести " << i << " число: ";           // вивести запрошення,
      cin >> x;                                         // ввести число, i,
      if (x%2==0 && x!=0) p *= x;                     // якщо число парне, перемножити його.
    }
    cout << "Добуток парних чисел: " << p << endl;
    system("pause>>void");
    return 0;
}
```

Результати виконання програми:

```
Ввести 1 число: 3
Ввести 2 число: 1
Ввести 3 число: -7
Ввести 4 число: 6
Ввести 5 число: 3
Ввести 6 число: -2
Ввести 7 число: 4
Ввести 8 число: 7
Ввести 9 число: 11
Ввести 10 число: -4
Ввести 11 число: -5
Ввести 12 число: 3
Добуток парних чисел: 192
```



У цієї програми є один суттєвий недолік. Якщо парних чисел немає, то результатом обчислення добутку буде виведено 1. Замість цього бажано вивести повідомлення, що парних чисел введено не було. Для цього слід організувати підрахунок кількості введених парних чисел. Якщо після циклу ця кількість становитиме 0, то слід вивести повідомлення. Для обчислення кількості слід оголосити окрему цілу змінну, значення якої спочатку є 0 (парних чисел ще нема). Якщо введене число є парне, кількість збільшиться на 1.

Текст програми:

```
#include <iostream>
using namespace std;
int main()
{
    setlocale(0, ".1251");
    int i, x, p=1, k=0;
    for(i=1; i<=12; i++)
    { cout << "Ввести " << i << " число: ";
```

```

    cin >> x;
    if (x%2==0 && x!=0)           // Якщо число парне,
    { p *= x;                      // обчислюватиметься добуток
      k++;                         // і кількість таких чисел.
    }
  }
}
if (k > 0) // Якщо кількість парних чисел є більше за 0, виведеться на екран
  cout << "Добуток парних чисел: " << p << endl; // добуток,
else      // інакше – виведеться повідомлення, що парних чисел немає.
  cout << "Парних чисел немає." << endl;
system("pause>>void");
return 0;
}

```

Результати виконання програми:

```

Ввести 1 число: 3
Ввести 2 число: 1
Ввести 3 число: -7
Ввести 4 число: 5
Ввести 5 число: 3
Ввести 6 число: -7
Ввести 7 число: 3
Ввести 8 число: 7
Ввести 9 число: 11
Ввести 10 число: -17
Ввести 11 число: -5
Ввести 12 число: 3
Парних чисел немає.

```

Приклад 4.40. Ввести шість дійсних чисел та визначити найбільше з них.

Розв'язок. Алгоритм пошуку максимального числа послідовності:

- 1) ввести перше число x ;
- 2) вважати, що воно є максимальним: $\text{max}=x$;
- 3) у циклі почергово вводити решту чисел. Кожне з введених чисел порівнювати зі значенням max і, якщо число x буде більшим за max , запам'ятати його значення як max :

```

    if (x>max) max=x;

```

Текст програми та блок-схема:

```

#include <iostream>
using namespace std;
int main()
{ setlocale(0, ".1251");
  double x, max;
  cout << "Ввести 1-ше число: ";
  cin >> x;

```

```

max=x;
for (int i=2; i<=6; i++)
{ cout << "Ввести " << i << " число ";
  cin >> x;
  if (x>max) max=x;
}
cout<< "Найбільше число: " << max << endl;
system("pause>>void");
return 0;
}

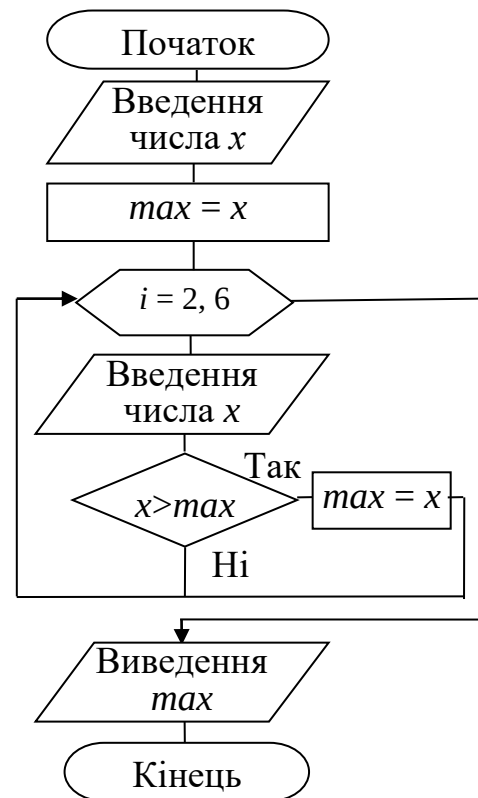
```

Результати виконання програми:

```

Ввести 1 число: 5.5
Ввести 2 число: 0.2
Ввести 3 число: -7
Ввести 4 число: 3.1
Ввести 5 число: 15.2
Ввести 6 число: -1.4
Найбільше число: 15.2

```



Приклад 4.41. Послідовно ввести вісім дійсних чисел й обчислити середнє арифметичне додатних з цих чисел.

Розв'язок. Для обчислення середнього арифметичного додатних чисел слід обчислити в циклі їхню суму та кількість. Після циклу, коли сума та кількість вже остаточно обчислено, значення суми слід поділити на кількість, якщо кількість є більше за нуль (серед введених чисел є додатні).

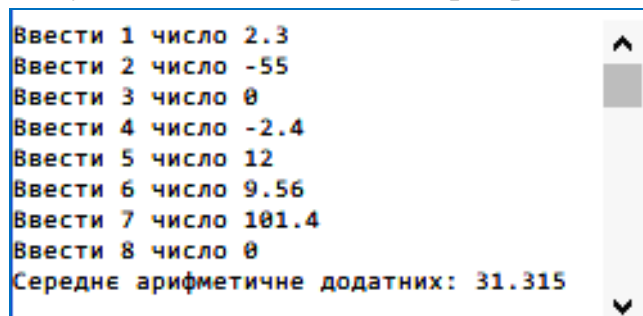
Текст програми:

```

#include <iostream>
using namespace std;
int main()
{
    system("color F0");
    setlocale(0, ".1251");
    int i, k = 0;    float x, s = 0, sred;
    for (i = 1; i <= 8; i++)
    { cout << "Ввести " << i << " число ";
      cin >> x;
      if (x > 0) {s += x; k++;} // Накопичення суми і кількості додатних
    }
    if (k > 0)
    { sred = s / k;
      cout << "Середнє арифметичне додатних: " << sred << endl;
    }
    else cout << "Додатних чисел нема" << endl;
    system("pause>>void");
    return 0;
}

```

Результати виконання програми:

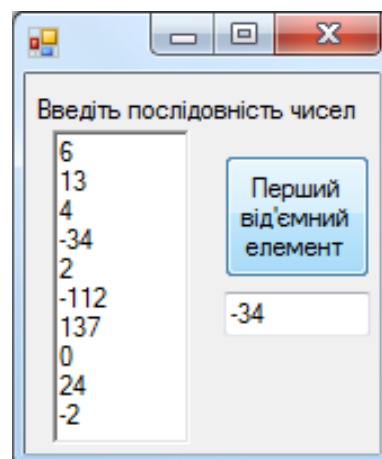


Приклад 4.42. Ввести послідовність цілих чисел і визначити перший від’ємний елемент.

Розв’язок. Для введення та програмного опрацювання довільної кількості чисел доцільно використати елемент `richTextBox1` у режимі створення програмних проектів Windows Forms. У програмі будуть опрацьовуватись лише заповнені рядки в `richTextBox1` (кількість рядків – властивість `Lines->Length`).

Текст програми та форма з результатами роботи:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    int v=0, x, n=richTextBox1->Lines->Length;
    for (int i=0; i<n; i++)
    {
        x=Convert::ToInt32(richTextBox1->Lines[i]);
        if (x<0) { v=x; break; }
    }
    textBox1->Text=v.ToString();
}
```



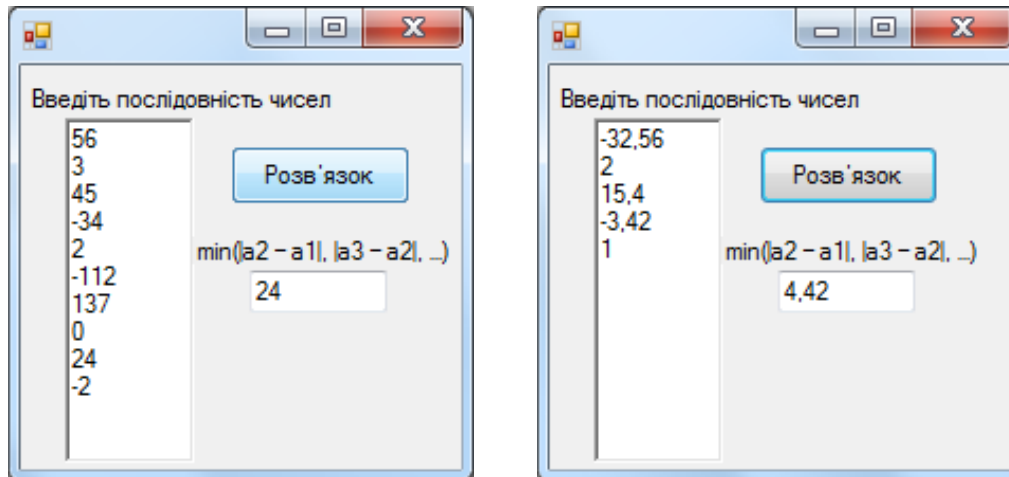
Приклад 4.43. Ввести послідовність дійсних чисел ($a_1, a_2, a_3, \dots$) й обчислити $\min(|a_2 - a_1|, |a_3 - a_2|, \dots)$.

Розв’язок. Уводитимемо числа до елемента `richTextBox`.

Текст програми:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    double min=1e3, x0, x=Convert::ToDouble(richTextBox1->Lines[0]);
    for (int i=1; i<richTextBox1->Lines->Length; i++)
    { x0=x;
      x=Convert::ToDouble(richTextBox1->Lines[i]);
      if (min > Math::Abs(x-x0)) min=Math::Abs(x-x0);
    }
    textBox1->Text=min.ToString();
}
```

Результати роботи:



Приклад 4.44. Ввести десять цілих чисел і визначити кількість непарних.

Текст програми та форма з результатами роботи:

// Кнопка «Кількість непарних»

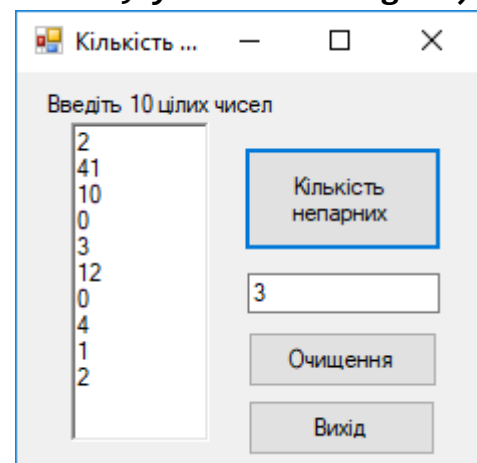
```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    int k=0, x;
    int n=richTextBox1->Lines->Length; // Кількість чисел
    if (n<10)
    { MessageBox::Show("Не всі 10 чисел введено"); return; }
    else
        if (n>10) MessageBox::Show("Введено більше 10 чисел.\n
                                   Оброблятися будуть тільки перші 10.");
    for (int i=0; i<10; i++)
    {
        x=Convert::ToInt32(richTextBox1->Lines[i]); // Зчитування чисел
        if (x%2==1) k++; // й обчислення кількості непарних
    }
    textBox1->Text=k.ToString(); // Виведення кількості непарних чисел
}
```

// Кнопка «Очищення»

```
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e)
{
    richTextBox1->Clear(); textBox1->Clear();
}
```

// Кнопка «Вихід»

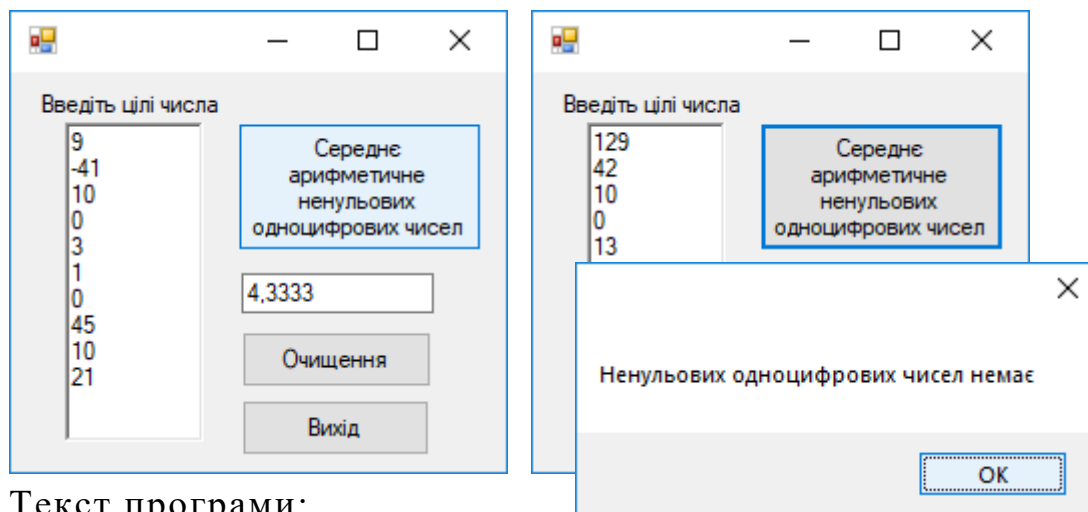
```
private: System::Void button1_Click
(System::Object^ sender, System::EventArgs^ e)
{
    Close();
}
```



Приклад 4.45. Ввести послідовність цілих чисел й обчислити середнє арифметичне ненульових одноцифрових чисел.

Розв'язок. Спочатку у циклі обчислимо суму та кількість ненульових одноцифрових чисел, а вже за циклом розділимо цю суму на кількість.

Для зчитування чисел з richTextBox використовуватимемо змінну *x*, для суми – змінну *sum*, для кількості – *k* і для середнього арифметичного – *aver*.



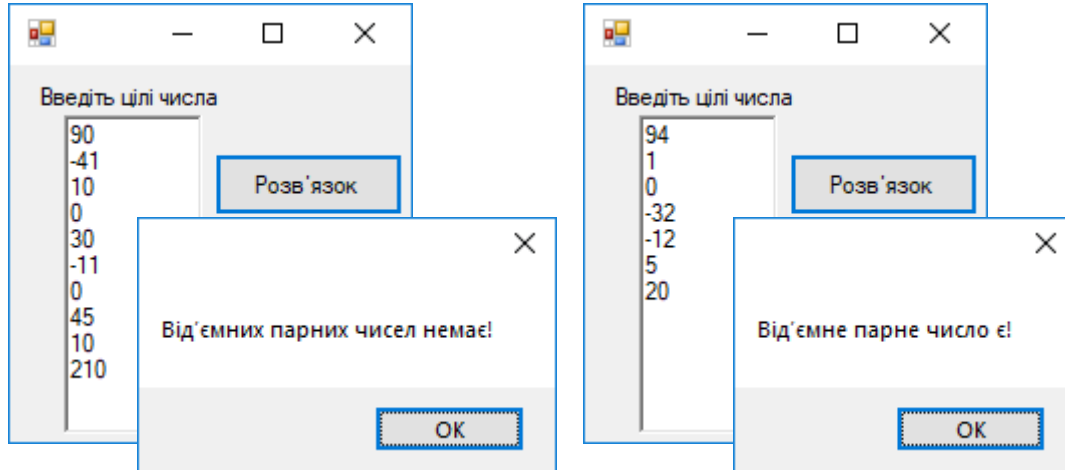
Текст програми:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    float aver, sum=0;    int x, k=0, i, n;
    n=richTextBox1->Lines->Length;    // Кількість чисел
    if (n==0)    // Якщо чисел у richTextBox1 немає, припинити обчислення
    { MessageBox::Show("Введіть числа!"); return; }
    for (i=0; i<n; i++) // Повторити n разів (послідовно для кожного числа)
    {
        x=Convert::ToInt32(richTextBox1->Lines[i]); // Зчитування чисел
        if (x>-10 && x<10 && x!=0)    // Якщо x – ненульове одноцифрове число,
        { sum+=x; k++; }    // додаємо його до суми sum і збільшуємо кількість на 1
    }
    if (k!=0)    // Якщо такі числа є,
    { aver=sum/k;    // обчислити середнє арифметичне,
      textBox1->Text=aver.ToString("0.####");
    }
    else    // інакше – вивести повідомлення
    { textBox1->Clear();
      MessageBox::Show("Ненульових одноцифрових чисел немає");
    }
}
```

Приклад 4.46. Ввести послідовність цілих чисел і визначити, чи є серед них від'ємні парні числа. Результат вивести у вигляді повідомлення.

Розв'язок. Для того щоб визначити, чи є в richTextBox від'ємні парні числа, можна обчислити їхню кількість, як це було зроблено у попередньому прикладі, і якщо ця кількість є більше за 0, можна стверджувати, що є як мінімум

одне таке число, інакше – таких чисел немає. Однак, якщо відшукали перше від’ємне парне число, немає потреби у перевірці решти чисел з `richTextBox`, можна одразу вивести повідомлення про результат і зупинити виконання програми за допомогою команди `return`. Якщо жодного від’ємного парного числа не віднайдено, то по завершенні циклу виконуватиметься наступна команда, яка виведе повідомлення про відсутність шуканих елементів.



Текст програми:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{ int x, i, n;
  n=richTextBox1->Lines->Length;    // Кількість чисел
  if (n==0)                          // Якщо чисел у немає, припинити обчислення
  { MessageBox::Show("Введіть числа!"); return;}
  for (i=0; i<n; i++)                // Повторити n разів (попередньо для кожного числа)
  { x=Convert::ToInt32(richTextBox1->Lines[i]); // Зчитування чисел
    if (x<0 && x%2==0)                // Якщо x – від'ємне парне число,
    { MessageBox::Show("Від'ємне парне число є!"); // вивести повідомлення
      return;                          // і зупинити виконання програми (вийти з функції)
    }
  }
  MessageBox::Show("Від'ємних парних чисел немає!");
}
```

Приклад 4.47. Написати програму, яка переведе введене користувачем десяткове число до двійкової системи числення.

Розв'язок. Алгоритм переведення цілого десяткового числа до двійкової системи числення полягає у послідовному діленні десяткового числа та його часток на значення основи системи 2. Ділення виконується, доки наступна частка не буде меншою за основу 2. Остача, обчислена на останньому кроці, є старшою (першою) цифрою переведеного числа. Результатом переведення такого числа до двійкової системи числення є запис останньої частки і всіх остач у зворотному порядку. Наприклад, переведення числа 133 з десяткової системи числення до двійкової виконується у такий спосіб:

$$\begin{array}{r}
 133 : 2 = 66 \text{ (1)} \\
 66 : 2 = 33 \text{ (0)} \\
 33 : 2 = 16 \text{ (1)} \\
 16 : 2 = 8 \text{ (0)} \\
 8 : 2 = 4 \text{ (0)} \\
 4 : 2 = 2 \text{ (0)} \\
 2 : 2 = 1 \text{ (0)}
 \end{array}$$

Результат: $133_{10} = 10000101_2$. Остачі від ділення записані в дужках після часток. Остання частка є старшою цифрою двійкового числа, до якого дописуються решта остач у порядку, зворотному до порядку її обчислення.

Текст програми в онлайн-компіляторі:

```

#include <iostream>
using namespace std;
int main()
{
    int dec; // Десяткове число
    int v, i; // Вага і номер формованого розряду
    printf("\nПереведення десяткового числа до двійкового \n");
    printf("Введіть ціле число від 0 до 255 і натисніть [Enter]");
    printf("\n-> ");
    scanf("%i", &dec);
    printf("Десятковому числу %i відповідає двійкове ", dec) ;
    v = 128; // Вага старшого (восьмого) розряду
    for (i = 1; i <= 8; i++)
    {
        if (dec >= v)
        {
            printf("1");
            dec -= v;
        }
        else printf("0");
        v = v / 2; // Вага наступного розряду удвічі менше
    }
    printf("\n\nДля завершення натисніть [Enter]");
    return 0;
}

```

Результати виконання програми:

```

Переведення десяткового числа до двійкового
Введіть ціле число від 0 до 255 і натисніть [Enter]
-> 200
Десятковому числу 200 відповідає двійкове 11001000
Для завершення натисніть [Enter]"

```

Приклад 4.48. Написати програму-тест для перевірки знань таблиці множення. Програма має вивести 10 прикладів і виставити оцінку: за 10 правиль-

них відповідей – «відмінно», за 9 та 8 – «добре», за 7 та 6 – «задовільно», за 5 і менш – «погано».

Текст програми:

```
#include <iostream>
#include <time.h>
using namespace std;

int main()
{
    system("color F0");
    setlocale(0, ".1251");
    // Програма перевіряє знання таблиці множення
    int numb1, numb2; // Оголошення змінних для зберігання значень співмножників,
    int res;          // добутку,
    int otv;          // відповіді,
    int kol = 0;      // кількості правильних відповідей,
    int i;            // лічильнику циклів
    time_t t;         // поточного часу – для ініціалізації генератора випадкових чисел
    printf("*** Перевірка знань таблиці множення ***\n");
    printf(" Введіть відповідь і натисніть [Enter]\n");
    srand((unsigned) time(&t)); // Ініціалізація генератора випадкових чисел
    for (i = 1; i <= 10; i++)    // 10 тестів
    {
        numb1 = rand()%7 + 2;    // Два випадкових числа
        numb2 = rand()%7 + 2;    // у межах від 2 до 9
        res = numb1 * numb2;
        printf(" %i x %i = ", numb1, numb2);
        scanf(" %i",&otv);
        if (otv == res) kol++;
        else printf(" Ви помилились! %i x %i = %i \n Спробуйте ще раз\n",
                    numb1, numb2, res);
    }
    printf("\n Правильних відповідей: %i\n", kol);
    printf(" Ваша оцінка: ");
    switch (kol)
    {
        case 10: puts(" Відмінно"); break;
        case 9:  puts(" Добре"); break;
        case 8:  puts(" Добре"); break;
        case 7:  puts(" Задовільно"); break;
        default: puts(" Погано"); break;
    }
    printf("\n Для завершення натисніть [Enter]");
    system("pause>>void");
    return 0;
}
```

Результати виконання програми:

*** Перевірка знань таблиці множення ***

Введіть відповідь і натисніть [Enter]

8 x 4 = 32

3 x 2 = 6

5 x 7 = 35

9 x 6 = 56

Ви помилились! 9 x 6 = 54

Спробуйте ще раз

6 x 7 = 42

9 x 3 = 27

8 x 8 = 64

4 x 3 = 12

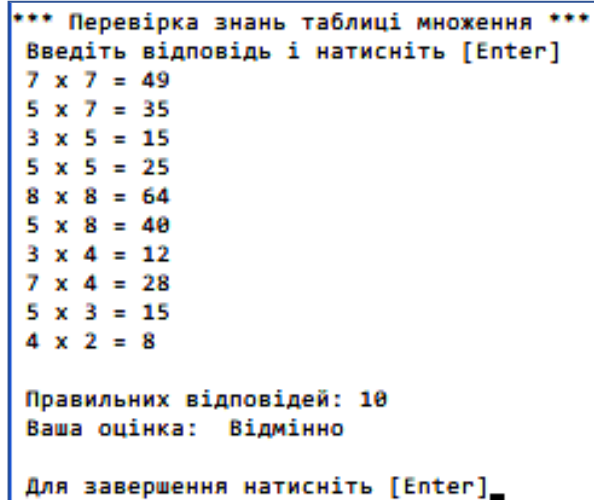
2 x 6 = 12

7 x 8 = 56

Правильних відповідей: 9

Ваша оцінка: Добре

Для завершення натисніть [Enter]



```

*** Перевірка знань таблиці множення ***
Введіть відповідь і натисніть [Enter]
7 x 7 = 49
5 x 7 = 35
3 x 5 = 15
5 x 5 = 25
8 x 8 = 64
5 x 8 = 40
3 x 4 = 12
7 x 4 = 28
5 x 3 = 15
4 x 2 = 8

Правильних відповідей: 10
Ваша оцінка: Відмінно

Для завершення натисніть [Enter]_

```

4.3.3 Вкладені цикли

Цикли може бути вкладено один в одного. При використанні вкладених циклів треба складати програму в такий спосіб, щоб внутрішній цикл повністю вкладався в тіло зовнішнього циклу, тобто цикли не повинні перетинатися. Своєю чергою, внутрішній цикл може містити власні вкладені цикли. Імена параметрів зовнішнього та внутрішнього циклів мають бути різними. Припускаються такі конструкції:

```

for (k=1; k<=10; k++)
{
    ...
    for (i=1; i<=10; i++)
    {
        ...
        for (m=1; m<=10; m++)
        {
            ...
        }
    }
}

```

Приклади проектів програм із вкладеними циклами

Приклад 4.49. Обчислити $S = \sum_{i=1}^m \frac{i}{i-2} \prod_{k=1}^{i+1} \frac{k+3}{k}$, значення m ввести з екрана. З

обчислень вилучити доданки та множники, які дорівнюють нулю в чисельнику чи то знаменнику.

Розв'язок. В цьому прикладі програми наведені цикли є вкладеними один в одного, позаяк параметр внутрішнього циклу k залежить від параметра зовнішнього циклу i (k змінюється від 1 до i).

Добуток $\prod_{k=1}^{i+1} \frac{k+3}{k}$ є співмножником доданка й обчислюється у внутрішньому циклі у змінній p . Оскільки внутрішній цикл складається лише з одного оператора, то операторні дужки $\{ \}$ не є обов'язковими.

Перед зовнішнім циклом для обчислення суми слід обнулити змінну S , в якій будуть накопичуватись доданки, а перед зовнішнім циклом для обчислення добутку змінній p слід присвоїти значення 1.

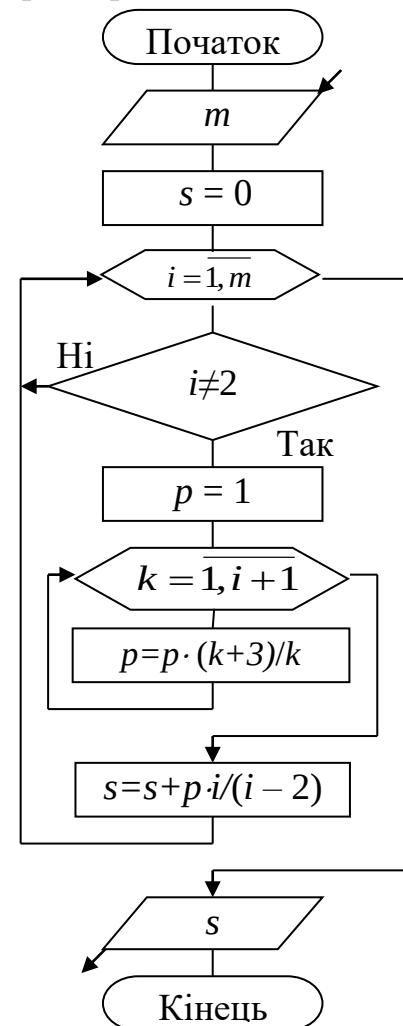
Позаяк при обчислюванні добутку беруть участь лише цілі числа, то, щоб при діленні не втратити дробову частину, слід перетворити чисельник до дійсного типу; для цього можна дописати крапку до числа 3: $(k+3.0)/k$.

Текст програми в онлайн-компіляторі:

```
#include <iostream>
#include <time.h>
using namespace std;
int main()
{
    int i, k, m;
    cout<<"Input m = "; cin>>m;
    float S = 0, p;
    for (i = 1; i <= m; i++)
        if ( i != 2)
        { p = 1;
          for (k = 1; k <= i+1; k++)
              p*=(k+3.0)/k;
          S += i/(i-2.0) * p;
        }
    cout<<"S = "<<S;
    return 0;
}
```

Результати виконання програми:

```
Input m = 4
S = 207
```



Приклад 4.50. Обчислити суму ряду $S = \sum_{i=1}^7 \frac{2x^{2i-1}}{3(2i-1)!}$, де $i = 1, 2, \dots, 7$.

Розв'язок. Для обчислення суми S треба підсумувати 7 доданків, для обчислення кожного з яких слід сформулювати вкладений цикл для підрахунку факторіалів $(2i-1)!$. У даній програмі кожний доданок обчислюється в окремій змінній a .

Текст програми в онлайн-компіляторі:

```
#include <iostream>
#include <math.h>
using namespace std;
```

```

int main()
{
    int i, k, fact; float s=0, a, x;
    cout<<" Input x = "; cin>>x;
    for (i=1; i<=7; i++)          // Зовнішній цикл
    { fact=1;
      for (k=1; k<=2*i-1; k++) // Внутрішній цикл для обчислення (2i-1)!
        fact *= k;
      a = 2*pow(x,2*i-1)/(3*fact);
      s += a;
    }
    cout<<" s = "<<s;
    return 0;
}

```

Результати виконання програми:

```

Input x = 2.4
s = 3.64426

```

Приклад 4.51. Написати програму, котра генерує три послідовності з десяти випадкових чисел в діапазоні від 1 до 10, виводить на екран й обчислює середнє арифметичне кожної послідовності.

Текст програми в онлайн-компіляторі:

```

#include <iostream>
#include <time.h>
using namespace std;
int main()
{ int r;          // Випадкове число
  int sum;        // Сума чисел послідовності
  float sr;       // Середнє арифметичне
  int i,j;        // Лічильники циклів
  time_t t; // Поточний час – для ініціалізації генератора випадкових чисел
  // Ініціалізація генератора випадкових чисел (див. докладніше підрозд. 3.6)
  srand((unsigned) time(&t));
  for (i = 1; i <= 3; i++) // Організуються три послідовності
  { printf("\n Випадкові числа: ");
    sum = 0;
    for (j = 1; j <= 10; j++) // Для послідовності з 10-ти чисел
    { r = rand() % 10 +1 ;    // генерується випадкове число
      printf("%i ", r) ;    // i виводиться його значення
      sum += r;
    }
    sr = (float)sum/10;
    printf("\n Середнє арифметичне: %3.2f\n", sr);
  }
  return 0;
}

```

Результат виконання програми:

Випадкові числа: 5 6 7 8 5 7 2 9 6 1

Середнє арифметичне: 5.60

Випадкові числа: 9 9 10 6 5 8 1 6 6 7

Середнє арифметичне: 6.70

Випадкові числа: 8 1 5 4 2 2 7 1 10 2

Середнє арифметичне: 4.20

Приклад 4.52. Вивести на екран квадрат Піфагора – таблицю множення.

Текст програми в онлайн-компіляторі:

```
#include <iostream>
using namespace std;
int main()
{
    int i,j;                // Номер рядка і стовпця таблиці
    printf("\n Квадрат Піфагора - таблиця множення \n\n\t");
    for (j=1; j<=10; j++) printf("\t%4i", j);
    printf("\n \n");
    for (i=1; i<=10; i++)
    { printf("%4i\t", i);
      for (j=1; j<=10; j++) printf("%4i\t", i*j);
      printf("\n");
    }
    return 0;
}
```

Результат виконання програми:

Квадрат Піфагора – таблиця множення

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

Приклад 4.53. Ввести цілі числа m і n та вивести на екран:

```
*****
*****
*****
```

де m – кількість рядків «зірочок» і n – кількість «зірочок» у рядку.

Розв'язок. Для розв'язання поставленого завдання слід організувати два вкладені цикли: зовнішній – для організації виведення окремих m рядків, тобто переходу на новий рядок, і внутрішній – для безпосереднього виведення n «зірочок» у рядок.

Текст програми в онлайн-компіляторі:

```
#include <iostream>
using namespace std;
int main()
{
    int i, j, m, n;
    cout << "Ввести кількість рядків m: ";
    cin >> m;
    cout << "Ввести кількість стовпців n: ";
    cin >> n;
    for (i=1; i<=m; i++)           // Цикл по рядках
    { for (j=1; j<=n; j++) cout << "*"; // Виведення рядка з n «зірочок»
      cout << endl;                // Перехід на новий рядок
    }
    return 0;
}
```

Результат виконання програми:

```
Ввести кількість рядків m: 2
Ввести кількість стовпців n: 8
*****
*****
```

Приклад 4.54. Ввести кількість рядків і вивести на екран трикутник «зірочок»:

```
*
**
***
****
. . .
```

Розв'язок. Як і у попередньому прикладі, для виведення одного рядка слід сформулювати окремий цикл, а для виведення кількох таких рядків необхідно вкласти перший цикл у цикл, який буде формувати перехід до наступного рядка. У першому рядку має бути лише одна «зірочка», у другому – дві, у третьому – три тощо. Тобто рядок з номером i має складатися з i «зірочок», тому внутрішній цикл виконуватиметься i разів.

Текст програми в онлайн-компіляторі:

```
#include <iostream>
using namespace std;
int main()
{
    int i, j, m;
```

```

cout << "Ввести кількість рядків m: ";
cin >> m;
for (i=1; i<=m; i++)
{ for (j=1; j<=i; j++) cout << "*";
  cout << endl;
}
return 0;
}

```

Результат виконання програми:

```

Ввести кількість рядків m: 5
*
**
***
****
*****

```

4.3.4 Оператори циклу з передумовою while та післяумовою do-while

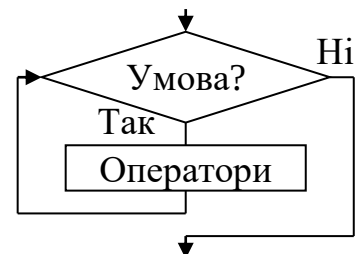
Оператори з передумовою та післяумовою використовуються для організації циклів і є альтернативними операторові **for**. Зазвичай цикл з передумовою використовується, якщо кількість повторювань заздалегідь є невідома або немає явно вираженого кроку змінювання параметра циклу. А для багаторазових повторювань тіла циклу відомим є вираз умови, за істинності якої цикл продовжує виконання. Цю умову слід перевіряти кожного разу перед черговим повторенням. Наприклад, при зчитуванні даних з файлу умовою є наявність цих даних у файлі, тобто повторювати читання даних слід аж допоки вказівник не дійде кінця файлу (докладніше див. розд. 12).

Синтаксис циклу з передумовою:

```

while (<умова>)
{ <тіло циклу> };

```



Послідовність операторів (тіло циклу) виконується, допоки умова є істинна (**true**, має ненульове значення), а вихід з циклу здійснюється, коли умова стане хибною (**false**, матиме нульове значення). Якщо умова є хибною при входженні до циклу, то послідовність операторів не виконуватиметься жодного разу, а керування передаватиметься до наступного оператора програми.

Цикл з післяумовою використовується, якщо є потреба перевіряти умову кожного разу після чергового повторення. Відмінність циклу з передумовою від циклу з післяумовою полягає в першій ітерації: цикл з післяумовою завжди виконується принаймні одноразово незалежно від умови.

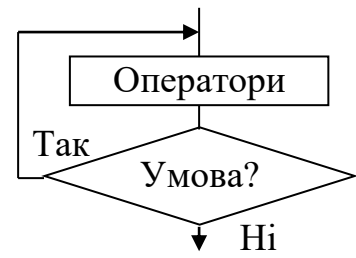
Синтаксис циклу з післяумовою:

```

do {
    <тіло циклу>
} while (<умова>);

```

Послідовність операторів (тіло циклу) виконується один чи кілька разів, допоки умова стане хибною (**false** чи дорівнюватиме нулю). Якщо умова є істинна (ненульова), то оператори тіла циклу виконуються повторно. Оператор циклу **do-while** використовується в тих випадках, коли є потреба виконати тіло циклу хоча б одноразово, оскільки перевірка умови здійснюється після виконання операторів.



Якщо тіло циклу складається з одного оператора, то операторні дужки {} не є обов'язкові.

Оператори **while** та **do-while** можуть завчасно завершитись при виконванні операторів **break** (див. п. 4.3.5), **goto** (див. п. 4.2.2), **return** (вихід з поточної функції, див. підрозд. 8.1) усередині тіла циклу.

Варто зауважити, що в тілі циклу слід передбачати змінювання параметрів, які беруть участь в умові, інакше умову виходу з циклу ніколи не буде виконано й відбуватиметься зациклювання.

Розглянемо відмінність роботи різних операторів циклу на прикладі обчислювання суми всіх непарних чисел у діапазоні від 10 до 100:

1) з використанням оператора **for**

```
int i, s=0;
for (i=11; i<100; i += 2) s += i;
```

2) з використанням оператора **while**

```
int s=0, i=11;
while (i<100) { s += i; i += 2; }
```

3) з використанням оператора **do-while**

```
int s=0, i=11;
do { s += i; i += 2;} while(i<100);
```

Приклади програм із застосуванням циклів **while** та **do-while**

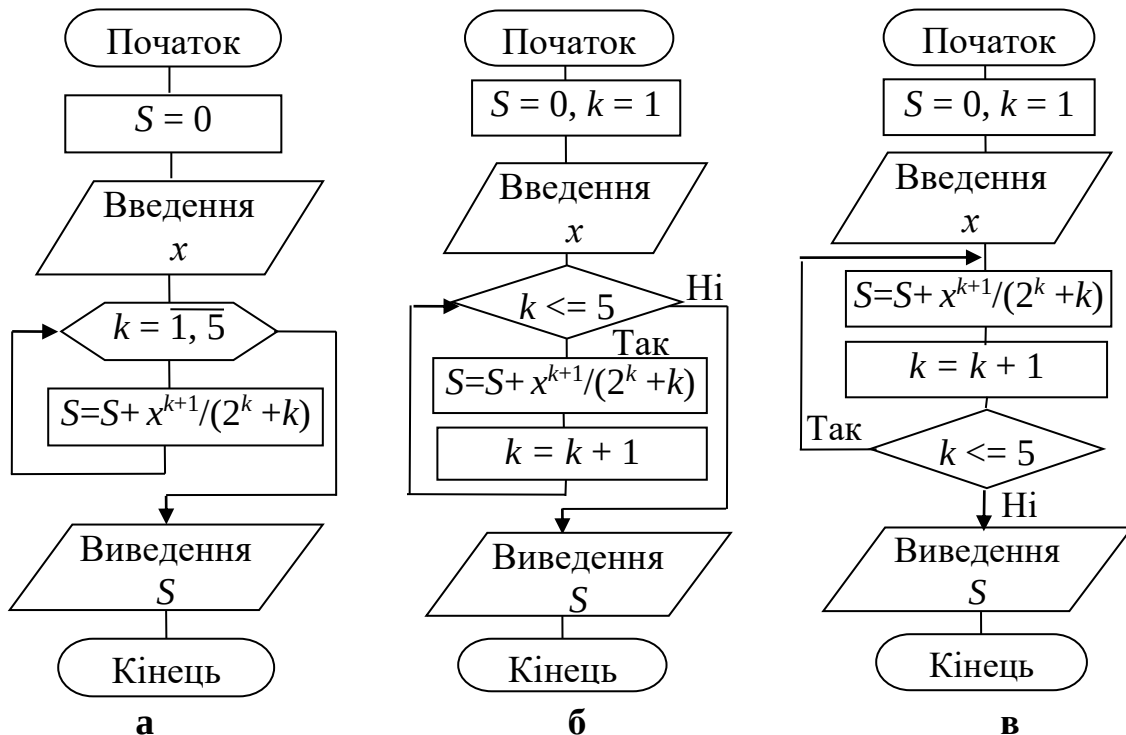
Приклад 4.55. Обчислити суму ряду $f(x) = \sum_{k=1}^5 \frac{x^{k+1}}{2^k + k}$ трьома варіантами,

використовуючи різні оператори циклу. Цей приклад продемонструє відмінності та специфіку використання різних операторів циклу в C++.

Розв'язок. Для наочності створимо програмний проект з формою, де буде вводиться значення x та виводитимуться результати, утворені трьома різними способами у трьох різних функціях.

Схеми алгоритмів усіх трьох програм наведено далі.

Схеми алгоритмів:



Схеми алгоритмів програм з різними операторами циклу:

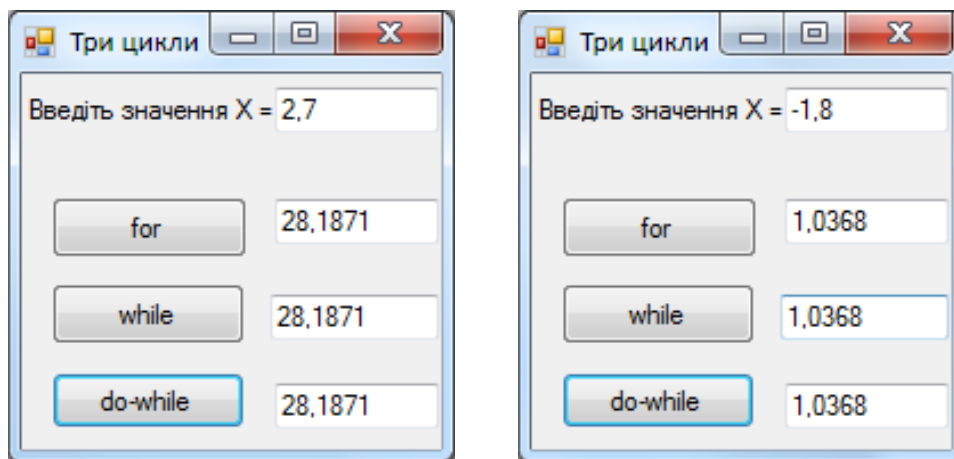
а) **for**; б) **while**; в) **do-while**

Програма з використанням різних операторів циклу:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{ double S = 0, x = Convert::ToDouble(textBox1->Text);
  for (int k=1; k<=5; k++)
    S += Math::Pow(x,k+1)/(Math::Pow(2,k)+k);
  textBox2->Text = S.ToString("0.0000");
}

private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e)
{ double S = 0, x = Convert::ToDouble(textBox1->Text);
  int k = 1;
  while (k <= 5)
  { S += Math::Pow(x,k+1)/(Math::Pow(2,k)+k);
    k++;
  }
  textBox3->Text = S.ToString("0.0000");
}

private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e)
{ double S = 0, x = Convert::ToDouble(textBox1->Text);
  int k = 1;
  do {
    S += Math::Pow(x,k+1)/(Math::Pow(2,k)+k);
    k++;
  }
  while (k <= 5);
  textBox4->Text = S.ToString("0.0000");
}
```



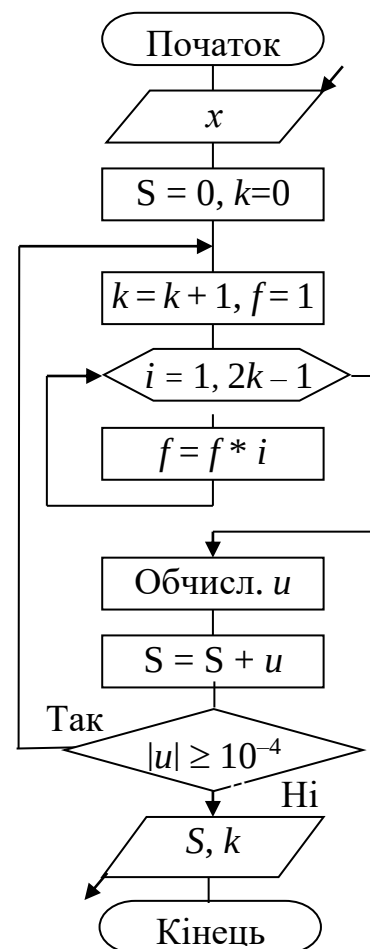
Приклад 4.56. Обчислити суму ряду $S = \sum_{k=1}^{\infty} \frac{x^{2k+1}}{3^k (2k-1)!}$, підсумовуючи

члени ряду, значення яких за модулем є більше за задану точність $\epsilon = 10^{-4}$. Визначити кількість доданків. Значення x ($-2 < x < 2$) вводити з клавіатури.

Розв'язок. Для цього завдання в програмі недоцільно використовувати оператор циклу з параметром **for**, позаяк кількість повторювань попередньо є невідома. Доцільним буде використання оператора циклу з післяумовою **do-while**, оскільки на момент першої перевірки умови вже треба знати значення першого доданка.

Текст консольної програми та блок-схема:

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{ setlocale(0, ".1251");
  double x, f, u, s=0;
  int i, k=0;
  cout << "Введіть значення x = "; cin >> x;
  cout << "\nРезультати:\n";
  do // Цикл з післяумовою
  { k++; // Збільшення змінної k на 1
    // Обчислення факторіалу
    for(i=1, f=1; i<=2*k-1; i++) f *= i;
    // Обчислення k-го доданка
    u=pow(x, 2*k+1)/(pow(3.,k)*f);
    s+=u; // Підсумовування доданків
  }
  while(fabs(u)>=1e-4);
  cout << "Сума = " << s << endl;
  cout << "Кількість доданків = " << k << endl;
  system("pause>>void");
  return 0;
}
```



Результати роботи:

Введіть значення $x = 2.5$

Результати:

Сума = 7.21478

Кількість доданків = 6

Приклад 4.57. Обчислити суму знакозмінного ряду $y = \sum_{k=1}^{\infty} \frac{(-1)^{k+1} x^{2k-1}}{(k+3)(2k)!}$, підсумовуючи члени ряду, значення яких за модулем є більше за задану точність ε . Визначити кількість доданків. Значення x ($-2 < x < 2$) і $\varepsilon = 10^{-4}$ вводити з клавіатури.

Розв'язок. Множник $(-1)^{k+1}$ за непарних $k = 1, 3, 5, \dots$ дорівнює 1, а за парних $k = 2, 4, \dots$ – дорівнює (-1) . Отже, поданий ряд є знакозмінним, де всі парні доданки будуть від'ємними, а всі непарні – зі знаком "+". Для наочності й контролю правильності у розв'язку окремо наведемо всі доданки.

Для того щоби зробити алгоритм програми більш оптимальним, його можна вдосконалити, але для цього слід обчислити рекурентний множник. Це дозволить у даній програмі позбавитись від вкладеного циклу для обчислення факторіалу й оператора перевірки на парність k .

```
u = x/8; s = u;
do
{
    k++;
    cout<< "доданок " << k << " : " << u << endl;
    r = -x*x*(k+2)/((3+k)*(2*k)*(2*k-1));
    u *= r;
    s += u;
}
while(fabs(u) >= eps);
```

Зупинимось більш докладно на виведенні рекурентної формули.

Подамо суму ряду $y = \sum_{k=1}^{\infty} \frac{(-1)^{k+1} x^{2k-1}}{(k+3)(2k)!}$ у вигляді $y = \sum_{k=1}^{\infty} u_k$, де

$u_k = \frac{(-1)^{k+1} x^{2k-1}}{(k+3)(2k)!}$. Рекурентний множник R – це співвідношення двох поряд розміщених членів ряду:

$$u_2 = R \cdot u_1, \quad u_3 = R \cdot u_2, \quad \dots, \quad u_k = R \cdot u_{k-1},$$

звідки

$$R = \frac{u_k}{u_{k-1}}.$$

Для визначення R у цю формулу слід підставити $u_k = \frac{(-1)^{k+1} x^{2k-1}}{(k+3)(2k)!}$ та u_{k-1} . Для

обчислення u_{k-1} підставимо у вираз для u_k $(k-1)$ замість k :

$$u_{k-1} = \frac{(-1)^k x^{2(k-1)-1}}{(k+2)(2k-2)!} = \frac{(-1)^k x^{2k-3}}{(k+2)(2k-2)!};$$

$$R = \frac{u_k}{u_{k-1}} = u_k = \frac{(-1)^{k+1} x^{2k-1}}{(k+3)(2k)!} \cdot \frac{(k+2)(2k-2)!}{(-1)^k x^{2k-3}} = \frac{(-1)^{k+1} \cdot x^{2k-1} \cdot (k+2) \cdot (2k-2)!}{(-1)^k \cdot x^{2k-3} \cdot (k+3) \cdot (2k)!} =$$

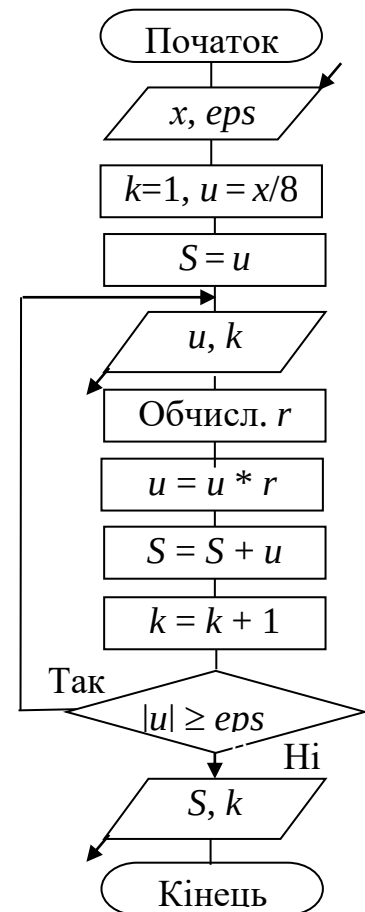
$$= \frac{(-1)^{k+1-k} \cdot x^{2k-1-(2k-3)} \cdot (k+2)}{(k+3)} \cdot \frac{1 \cdot 2 \cdot \dots \cdot (2k-2)}{1 \cdot 2 \cdot \dots \cdot (2k-2) \cdot (2k-1) \cdot 2k} = -\frac{x^2(k+2)}{2k(k+3)(2k-1)}.$$

Крім рекурентного множника R , треба обчислити перший член ряду за $k = 1$:

$$u_1 = \frac{(-1)^2 x}{4 \cdot (2)!} = \frac{x}{8}.$$

Текст консольної програми та блок-схема:

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
    setlocale(0, ".1251");
    double x, u, r, s, eps; int k=1;
    cout<<"Введіть значення x = "; cin>> x;
    cout<<"Введіть значення точності "; cin>> eps;
    cout<<"\nРезультати:\n";
    u=x/8; s=u;
    do
    {
        cout<<"доданок "<< k <<": " << u << endl;
        r = -x*x*(k+2)/((3+k)*(2*k) *(2*k-1));
        u *= r; s += u;
        k++;
    }
    while(fabs(u) >= eps);
    cout << "Сума = "<< s << endl;
    cout << "Кількість доданків = "<< k << endl;
    system("pause>>void");
    return 0;
}
```



Результати роботи:

```
Введіть значення x = 1.4
Введіть значення точності 0.0001
Результати:
доданок 1: 0.175
доданок 2: -0.128625
доданок 3: 0.016807
доданок 4: -0.000915048
Сума = 0.0622944
```

Приклад 4.58. Визначити і вивести всі дільники введеного натурального числа.

Текст програми:

```
#include <iostream>
int main()
{ system("cls");           // Очистити екран
  system("chcp 1251");
  system("color F0");       // Встановити білий фон і чорний текст
  unsigned int x, i;
  std::cout << "Введіть x: "; std::cin >> x;
  std::cout << std::endl << "Дільники: ";
  for (i=1; i<=x; i++)
    if (x%i == 0)           // Якщо ділиться без остачі,
      std::cout << i << "\t"; // вивести черговий дільник
  std::cin.get();
  std::cin.get();
  return 0;
}
```

Введіть x: 45

Дільники: 1 3 5 9 15 45

Введіть x: 121

Дільники: 1 11 121

Приклад 4.59. Визначити кількість цифр введеного натурального числа.

Текст програми:

```
#include <iostream>
using namespace std;
int main()
{ unsigned int N, kol;
  cout << "N="; cin >> N;
  for (kol=1; N/10>0; kol++,N/=10);
  cout << "kol=" << kol << endl;
  system("pause>>void");
  return 0;
}
```

N=50026
kol=5

Приклад 4.60. Визначити найменшу та найбільшу цифру введеного натурального числа.

Текст програми:

```
#include <iostream>
using namespace std;
int main()
{ unsigned int N, z, min, max;
  cout<<"N="; cin>>N;
  for (max=min=N%10; N>0; N/=10)
  { z=N%10;
    if (z>max) max=z;
    if (z<min) min=z;
  }
  cout << "min=" << min << endl << "max=" << max << endl;
  system("pause>>void");
  return 0;
}
```

N=550308
min=0
max=8

Приклад 4.61. Перевірити чи є введене натуральне число паліндромом.

Паліндром – це число (слово чи фраза), яке однаково читається в обох напрямках, або, інакше кажучи, будь-який симетричний щодо своєї середини набір символів. Наприклад, числа 404, 12521 – паліндроми.

Текст програми:

```
#include <iostream>
using namespace std;
int main()
{ setlocale(0, ".1251");
  unsigned long long n,m,d; // int kol, i; bool FL;
  cout<<"N="; cin>>n;
  d=0; m=n;
  while (m!=0)
  { d = d*10 + m%10;
    m /= 10;
  }
  if (d==n) cout<<"Так, число є паліндромом"<<endl;
  else cout<<"Ні, число не є паліндромом "<<endl;
  system("pause>>void");
  return 0;
}
```

```
N=25052
Так, число є паліндромом
```

Приклад 4.62. Обчислити квадрати натуральних чисел від 1 до 10 трьома варіантами, використовуючи різні оператори циклу.

Розв'язок. Оскільки натуральні числа – це цілі додатні числа 1, 2, 3, 4, ..., доцільно використати змінну *i* як параметр циклу, яка змінюватиметься від 1 до 10.

У циклі десятикратно повторюватимуться такі дії:

- ✓ обчислення квадрату числа *i*;
- ✓ виведення на екран значення змінної *i* та її квадрату;
- ✓ збільшення змінної *i* на 1.

Текст програми з циклом **for** в онлайн-компіляторі:

```
#include <iostream>
using namespace std;
int main()
{ int i, x;
  for (i=1; i<=10; i++)
  { x = i * i;
    cout<<"Квадрат числа "<<i<<" дорівнює "<< x << endl;
  }
  return 0;
}
```

```
Квадрат числа 1 дорівнює 1
Квадрат числа 2 дорівнює 4
Квадрат числа 3 дорівнює 9
Квадрат числа 4 дорівнює 16
Квадрат числа 5 дорівнює 25
Квадрат числа 6 дорівнює 36
Квадрат числа 7 дорівнює 49
Квадрат числа 8 дорівнює 64
Квадрат числа 9 дорівнює 81
Квадрат числа 10 дорівнює 100
```

Напишемо цю програму з циклами `while` та `do-while` замість `for`. Перше значення змінної `i` слід задати до циклу. Також слід пам'ятати, що умовні оператори циклу не збільшують автоматично значення параметра, а тому збільшення параметра слід прописати у тілі циклу.

Текст програми з циклом `while` в онлайн-компіляторі:

```
int main()
{ int i=1, x;
  while (i<=10)      // Якщо число не перевищує 10,
  { x = i * i;        // обчислюватиметься квадрат його значення
                      // і виводитиметься число разом із квадратом
    cout << "Квадрат числа " << i << " дорівнює " << x << endl;
    i++;              // Збільшення змінної циклу на 1
  }
  return 0;
}
```

Текст ідентичної програми з циклом `do-while` в онлайн-компіляторі:

```
int main()
{ int i=1, x;
  do
  { x = i * i;
    cout << "Квадрат числа " << i << " дорівнює " << x << endl;
    i++;
  } while(i<=10);
  return 0;
}
```

Приклад 4.63. Вводити цілі числа, допоки не буде введене трицифрове число, й обчислити кількість введених чисел та суму чисел, які належать до інтервалу $[-4, 11)$.

Розв'язок. Трицифрове число – це число, модуль якого перебуває поміж 100 та 999. Отже, вираз умови продовження виконання циклу матиме вигляд:

$$\text{abs}(x) < 100 \quad || \quad \text{abs}(x) > 999$$

Якщо для розв'язання поставленого завдання застосовувати цикл `while`, то ще до початку циклу слід знати перше значення змінної `x`, яка буде вводитись згодом у циклі, або присвоїти `x` будь-яке нетрицифрове значення до циклу, наприклад, 0, щоб можна було увійти до циклу.

Текст програми з циклом `while`:

```
#include <iostream>
using namespace std;
int main()
{
  system("color F0");
  setlocale(0, ".1251");
```

```
int x=0, k=0, s=0;
while(abs(x)<100 || abs(x)>999) // Допоки число x не є трицифрове,
{ cout << " Ввести число: ";  cin >> x;
  k++;                          // збільшується кількість введених чисел i,
  if (x>=-4 && x<11)            // якщо x належить до проміжку [-4, 11),
    s += x;                    // додається його значення до суми
}
cout << " Введено чисел: " << k << endl;
cout << " Сума чисел з проміжку [-4, 11): " << s << endl;
system("pause>>void");
return 0;
}
```

Результат виконання програми:

```
Ввести число: 4
Ввести число: 8
Ввести число: -1
Ввести число: 3
Ввести число: 9
Ввести число: 54
Ввести число: 2
Ввести число: -87
Ввести число: 4
Ввести число: 443
Введено чисел: 10
Сума чисел з проміжку [-4, 11): 29
```

У цьому прикладі більш оптимально використовувати цикл **do-while**, що надасть можливість не хвилюватися стосовно першого значення x – цикл розпочнеться поза всяких умов.

Текст основної програми:

```
int main()
{
  system("color F0");
  setlocale(0, ".1251");
  int x, k=0, s=0;
  do
  { cout << " Ввести число: ";  cin >> x;
    k++;
    if (x>=-4 && x<11) s += x;
  }
  while (abs(x)<100 || abs(x)>999);
  cout << " Введено чисел: " << k << endl;
  cout << " Сума чисел з проміжку [-4, 11): " << s << endl;
  system("pause>>void");
  return 0;
}
```

Приклад 4.64. Генерувати й виводити випадкові числа з проміжку $[-5, 10)$, допоки числа збільшуються. Обчислити середнє арифметичне цих чисел.

Розв'язок. Для генерування випадкових цілих додатних чисел існує функція `rand()` (див. докладніше підрозд. 3.6). Наприклад, для генерування чисел з проміжку $[-5, 10)$ слід записати цю функцію в такий спосіб:

```
x = rand()%15 - 5;
```

Числа мають зростати, тобто кожне наступне має бути більше за попереднє. Ми повинні зберігати попереднє число у певній змінній.

Текст основної програми:

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;
int main()
{
    system("color F0");
    setlocale(0, ".1251");
    int x, y, s, k = 1; time_t t;
    srand((unsigned) time(&t)); // Ініціалізація генератора випадкових чисел
    x = rand()%15 - 5;          // Перше випадкове число x
    cout<<" Число: "<<x<<endl;
    s = x;
    do { y = x; // У циклі запам'ятовується число x у змінній y,
        x = rand()%15 - 5; // і генерується наступне випадкове число x,
        cout<<" Число: "<<x<<endl; // яке виводиться на екран.
        if (x>y) // Порівнюється число з попереднім і, якщо
        { s+=x; // воно є більше за попереднє, обчислюється сума s
          k++; // та кількість k таких чисел.
        }
    }
    while(x>y); // Вихід з циклу відбудеться за умови
    // введення порівняно меншого числа.
    cout << "Кількість чисел: " << k << endl;
    cout << "Середнє арифметичне: " << (float) s/k << endl;
    system("pause>>void");
    return 0;
}
```

Результати виконання програми:

```
Число: -3
Число: 0
Число: 8
Число: 7
Кількість чисел: 3
Середнє арифметичне: 1.66667
```

Приклад 4.65. Написати програму обчислення найбільшого спільного дільника (НСД) двох цілих чисел, тобто реалізувати алгоритм Евкліда.

Текст програми:

```
#include <iostream>
using namespace std;
int main()
{ system("color F0");
  setlocale(0, ".1251");
  printf("\nОбчислення найбільшого спільного дільника");
  printf("двох цілих чисел.\n");
  printf("Введіть в один рядок два числа і натисніть <Enter>");
  printf("\n -> ");
  int nod;          // Найбільший спільний дільник (НСД)
  int n1, n2;        // Числа, НСД яких треба обчислити
  int r;             // Остача від ділення n1 на n2
  scanf("%i %i", &n1, &n2) ;
  printf("НСД чисел %i і %i - це ", n1, n2);
  while (n1 % n2)
  { r = n1 % n2; // Остача від ділення
    n1 = n2;
    n2 = r;
  }
  nod = n2;
  printf("%i\n", nod);
  system("pause>>void");
  return 0;
}
```

Результати виконання програми:

Обчислення найбільшого спільного дільника двох цілих чисел.

Введіть до одного рядка два числа і натисніть [Enter]

-> 121 33

НСД чисел 121 і 33 – це 11

Приклад 4.66. Ввести цілі числа до richTextBox1 і скопіювати парні числа з richTextBox1 до richTextBox2 допоки їхня сума не стане більшою за 10.

Розв'язок. При копіюванні чисел слід зупинитись, якщо вже скопійовано всі числа чи то якщо сума скопійованих чисел є більша за 10.

Наведемо три варіанти розв'язання, використовуючи три цикли: **for**, **while**, **do-while**.

Текст програми:

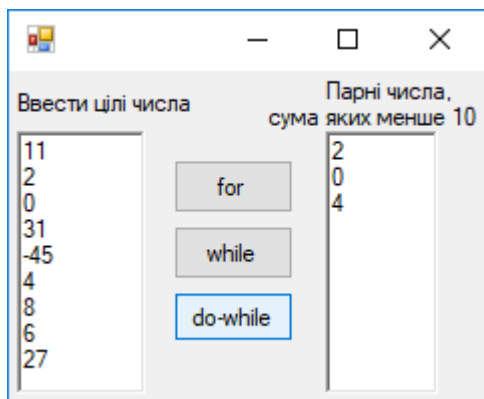
```
// Розв'язання за допомогою циклу for
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
  int x, i, n, S=0;
```

```

n=richTextBox1->Lines->Length;    // Кількість чисел
if (n==0)                          // Якщо чисел у немає, припинити обчислення
{ MessageBox::Show("Введіть числа!"); return;}
richTextBox2->Clear();
// У циклі умова i<n означає, що в richTextBox1 ще є числа для зчитування і копіювання
for (i=0; i<n; i++)
{
    x=Convert::ToInt32(richTextBox1->Lines[i]); // Зчитування числа
    if (x%2==0)                                // Якщо x - парне число
    { S += x;
      if (S<=10)                               // Якщо сума менша 10
        richTextBox2->Text += x.ToString()+"\n";
    }
}
}
// Розв'язок за допомогою циклу while
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e)
{
    int x, i=0, n, S=0;
    n=richTextBox1->Lines->Length;
    if (n==0) { MessageBox::Show("Введіть числа!"); return;}
    richTextBox2->Clear();
    while(i<n)
    {
        x=Convert::ToInt32(richTextBox1->Lines[i]);
        if (x%2==0)
        { S += x;
          if (S<=10) richTextBox2->Text += x.ToString()+"\n";
        }
        i++;
    }
}
// Розв'язок за допомогою циклу do-while
private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e)
{
    int x, i=0, n, S=0;
    n=richTextBox1->Lines->Length;
    if (n==0) { MessageBox::Show("Введіть числа!"); return;}
    richTextBox2->Clear();
    do
    { x=Convert::ToInt32(richTextBox1->Lines[i]);
      if (x%2==0)
      { S += x;
        if (S<=10) richTextBox2->Text += x.ToString()+"\n";
      }
      i++;
    } while(i<n);
}

```

Форма з результатами роботи наведеної програми:



Приклад 4.67. Зчитати числа з richTextBox, доки не зустрінеться число зі значенням 0. Віднайти найменше зі зчитаних чисел.

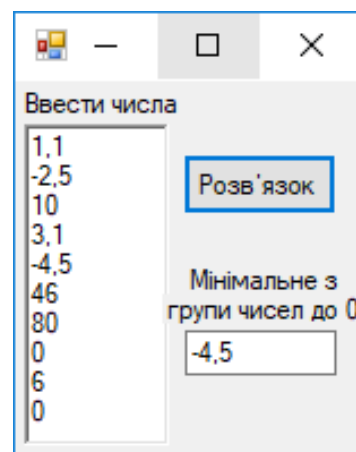
Розв'язок. Для пошуку мінімального числа використовуватимемо такий алгоритм:

- 1) поточному мінімуму присвоїти перше число;
- 2) пройти всіма числами і, якщо зустрінеться число, більше за поточний мінімум, присвоїти це число поточному мінімуму.

Після того, як всі числа переглянуто, поточний мінімум можна вважати за остаточний мінімум.

Текст програми:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    int i=0, n=richTextBox1->Lines->Length;
    double x, min;
    x=Convert::ToDouble(richTextBox1->Lines[0]);
    // Вважатимемо, що перше число - мінімальне
    min = x;
    while (x != 0 && i < n)
    { i++;
      x=Convert::ToDouble(richTextBox1->Lines[i]);
      if (x < min) // Якщо віднайшли менше число,
        min = x; // записати це число у min
    }
    textBox1->Text=min.ToString();
}
```



Приклад 4.68. Зчитати числа з richTextBox, доки не зустрінеться 0. Віднайти мінімальне додатне число.

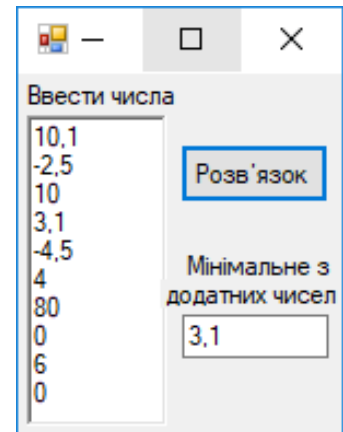
Розв'язок. Відмінність цього прикладу від попереднього полягає в тому, що мінімальне число треба віднайти з-посеред додатних чисел і присвоїти поточному мінімуму перше число не можна, оскільки воно може бути від'ємним.

Розглянемо два способи розв'язання цього завдання.

Перший спосіб

Спочатку пройдемо по числах в richTextBox1 і відшукаємо перше додатне число (за допомогою циклу **do-while**). Після цього пройдемо по решті чисел і віднайдемо мінімальне з додатних.

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    int i=0, n=richTextBox1->Lines->Length;
    double x, min;
    if (n==0) { MessageBox::Show("Введіть числа!"); return; }
    do
    { x=Convert::ToDouble(richTextBox1->Lines[i]);
      i++;
    }
    while (x<=0 && i<n);
    min = x;
    i=0;
    while (x != 0 && i<n)
    { x=Convert::ToDouble(richTextBox1->Lines[i]);
      if (x>0 && x<min) min = x;
      i++;
    }
    textBox1->Text=min.ToString();
}
```



Другий спосіб

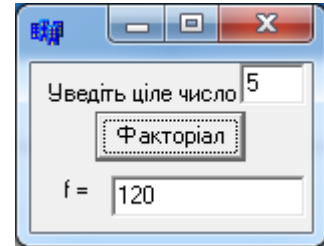
Спочатку присвоюємо змінній min значення 0. Зчитуємо числа з richTextBox1. Якщо число є від'ємним, зігноруємо його. Якщо число є додатним та min=0, це означає, що ще не зустрічались додатні числа, це – перше і його слід присвоїти min. Якщо число є додатним і min≠0, це означає, що додатні числа вже зустрічались і min вже має певне додатне значення (тимчасовий мінімум). У цьому разі порівнюємо число з min і, якщо воно є менше за min, присвоюємо його.

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    int i=0, n=richTextBox1->Lines->Length;
    double x, min=0;
    if (n==0) { MessageBox::Show("Введіть числа!"); return; }
    do
    { x=Convert::ToDouble(richTextBox1->Lines[i]);
      if (x>0)
        if (min==0 || x<min) min = x;
      i++;
    }
    while(x != 0 && i<n);
    textBox1->Text=min.ToString();
}
```


Приклад 4.69. Написати програму обчислення факторіала введенного цілого числа за допомогою оператора **while**.

Розв'язок. У порівнянні з обчисленням факторіала оператором **for** (див. п. 4.3.2) чи за допомогою рекурсивної функції (див. підрозд. 8.5) наведене розв'язання є самим лаконічним і елегантним.

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{ int f = 1, n;
  n = Convert::ToInt32(textBox1->Text);
  while ( n >= 1 ) f *= n--;
  textBox2->Text = f.ToString();
}
```



4.3.5 Оператори переривання виконання

Для завчасного переривання повторювань операторів циклу будь-якого типу в тілі циклу можна застосовувати оператор **break**, який перериває виконання оператора, в якому розміщено цей **break**, і передає керування на наступний оператор.

Наведемо приклад застосування оператора **break** на програмному фрагменті обчислення у змінній **k** кількості різних (неоднакових) елементів для масиву **a** з **m** елементів. Кожен черговий елемент **a[i]** порівнюється з наступними елементами масиву. Якщо він не збігається з жодним із цих елементів, лічильник **k** збільшується на одиницю. Інакше, внутрішній цикл переривається оператором **break** і починається нова ітерація зовнішнього циклу.

```
int k=1;
for (int i=0; i<m-1; i++)
{ for (int j=i+1; j<m; j++)
    if (a[i]==a[j]) break;
    if (j==m) k++;
}
```

Зауважимо, що всередині вкладених операторів **do-while**, **for**, **while** чи **switch** оператор **break** завершує лише самий внутрішній з цих операторів, тому **break** не можна використовувати для виходу з декількох вкладених циклів. Навіть повторення підряд двох операторів **break** не забезпечить вихід з вкладених циклів у такому фрагменті:

```
for (i=0; i<100; i++)
for (j=0; j<100; j++)
{ . . .
    if (a[i][j]<0) { break; break; }
    . . .
}
```

Так після виконання умови **a[i][j]<0** відбудеться вихід лише з внутрішнього циклу по змінній **j**, а виконання зовнішнього циклу по змінній **i** продовжиться, незважаючи на те, що оператор **break** записано двічі.

По своїй суті оператор **break** є оператором переходу і оператори записанні після нього виконуватись не будуть, якщо тільки їм не буде передано керування за допомогою інших операторів переходів.

Для виходу з обох циклів можна застосувати додаткову змінну, яка буде набувати ненульового значення лише за потреби виходу із зовнішнього циклу:

```
for (i=0; i<100; i++)
{ for (br2=j=0; j<100; j++)
  { . . .
    if ( a[i][j]<0 ) { br2=1; break;}
    . . .
  }
  if (br2) break;
  . . .
}
```

У наведеному прикладі відразу після завершення внутрішнього циклу перевірятиметься умова завершення зовнішнього циклу.

Крім того, передавати керування за межі вкладеної структури, можна використовуючи оператори **return** (вихід з поточної функції) (див. підрозд. 8.1) та **goto** (безумовний перехід) (див. п. 4.3.2).

Для переходу до наступної ітерації циклу можна застосовувати оператор **continue**. Цей оператор, подібно до оператора **break**, використовується лише всередині операторів циклу **for**, **while**, **do-while**, але, на відміну від останнього, оператор **continue** не припиняє подальше виконання циклу, а переходить до чергової ітерації того циклу, у тілі якого він виявився. Він як би імітує безумовний перехід на кінцевий оператор циклу, але не за межі самого циклу.

Оператор **continue**, так само як і оператор **break**, перериває найглибший з циклів.

```
for (int a = 1, b = 0; a < 100; b += a, a++)
{ if (b % 2) continue;
  . . .
}
```

// Опрацювання парних сум

У вищенаведеному прикладі оператор **continue** передає керування на чергову ітерацію циклу **for** за умови, коли сума чисел від 1 до 100 є непарною, не виконуючи операторів опрацювання парних сум.

Наведемо ще один програмний фрагмент з застосуванням оператора **continue** для обчислення суми додатних елементів масиву **a** з **m** елементів:

```
int s=0;
for (int i=0; i<m; i++)
{ if (a[i]<=0) continue; // Пропустити елемент
  s+=a[i];
}
```

У наведеному прикладі оператор **continue** використовується для пропускання від'ємних елементів масиву, підсумовуючи лише додатні.

Питання та завдання для самоконтролю

- 1) Який процес називають лінійним?
- 2) Який процес називають розгалуженням?
- 3) Які оператори в C++ використовуються для організації розгалужень?
- 4) Перелічити базові логічні операції.
- 5) Вказати послідовність дій при виконанні оператора та значення w :
`bool w=2*5<=17%3;`
- 6) Яким буде результат виконання логічного виразу $(7+3 > 16-4*3)$?
- 7) Обчислити логічний вираз $(-3 >= 5) \ || \ (7 < 9) \ \&\& \ (0 < 3)$.
- 8) Вказати значення w після виконання оператора: `bool w=2*5<=17%3;`
- 9) Засобами умовного оператора `if` скороченої і повної форми, а також

умовної операції $?:$ записати три варіанти обчислення $y = \begin{cases} \sin x^2 & \text{за } x > 0.5; \\ \cos^2 x & \text{за } x \leq 0.5. \end{cases}$

- 10) Назвати оператор без помилок:
 - а) `if(x<=6)y=2*x; else y=cos(x);` в) `if(a<>0) if(b<>0) y=2*x;`
 - б) `if y<=x then y:=exp(x*y);` г) `if(x>0)y=ln(x) else y=x;`
- 11) Вказати значення x після виконання фрагментів програми:
 - а) `float x=1.5;`
`if(x<=0.5) x=7.7;`
 - б) `float x=1.5;`
`if(x<=0.5) x=7.7; else x=3;`
- 12) Вказати значення змінної z після виконання операторів:


```
x=2.5;
if(x>=0.5) z=7.7; else z=5.5;
```
- 13) Вказати значення змінної f після виконання операторів:


```
f = 1; n = 3; i = 2;
M1: if (i > n) goto PP;
    f = f * i; i++; goto M1;
PP : ;
```
- 14) Вказати значення y після виконання фрагментів програми:
 - а) `float y=0; int n=1;`
`switch (n)`
`{case 1: { y=n/4.; break; }`
`case 2: { y=n*n; break; }`
`case 3: { y=n; break; } }`
 - б) `float y=0; int n=3;`
`switch (n)`
`{case 1: { y=n/4.; }`
`case 3: { y=n*n; }`
`case 5: { y=n+1; } }`
 - в) `float y=0; int n=4;`
`switch (n)`
`{case 2: { y=n/4.; break; }`
`case 5: { y=n*n; break; }`
`case 9: { y=n; break; } }`
 - г) `float y=0; int n=1;`
`switch (n)`
`{case 1: { y=n/4; }`
`case 3: { y=n*n; }`
`case 5: { y=n+1; } }`
- 15) У наведеному фрагменті програми


```
int nom = pow(2, 3);
switch (nom)
{ case 2 : y=d; break;
  case 8 : y=d*exp(x); break;
  case 10 : y=d*x; break;
}
```

оператор `switch` обчислюватиме вираз ... (записати увесь вираз).

16) Який процес називають циклічним?

17) Які оператори циклу використовуються у мові C++?

18) Скільки разів виконуватиметься оператор у циклі, тобто вказати значення `s`:

```
for(int k = -1, s=0; k <= 5; k++) s++;
```

19) Вказати помилки в таких фрагментах програм:

а) `int k, m=2, n=3;`

б) `int n=-7, m=2;`

```
for(k=1; k<=n; k++)n=n+m;
```

```
for(int k=n; k<=m; k--)k++;
```

20) Вказати значення `m` після виконання фрагментів програми:

а) `int k, m=1;`

б) `int m=1, n=5;`

```
for(k=1; k<=5; k++)m++;
```

```
for(int k=n; k>=1; k--)m*=k;
```

21) Назвати правильну, на Ваш погляд, послідовність номерів, для запису елементів оператора циклу `while`:

а) логічний вираз умови;

б) оператори тіла циклу;

в) `while`.

22) Вказати значення `n` після виконання фрагментів програми:

а) `int k=0, n=17;`

б) `int m=1, n=1;`

```
while(k<7){ k++; n--;}
```

```
while(m<5){ m+=2; n=n*m;}
```

23) Якими є структура і порядок виконання оператора циклу `do-while`?

24) Вказати значення `y` після виконання фрагментів програми:

а) `int i=1, y=1;`

б) `int k=1; float y=0;`

```
do {y*=i++;} while(y<7);
```

```
do{k+=2;y+=1./k;}while(k<5);
```

25) Записати трьома операторами циклу варіанти обчислення $S = \sum_{i=1}^6 i^2$.

26) Вказати значення `s` після виконання операторів:

```
s=0.5; i=0;
```

```
while(i<5) i++;
```

```
s+=1.0/i;
```

27) Назвати номер фрагмента програми з вкладеним циклом:

а) `for(k=1;k<=10;k++)` б) `for(k=1;k<=10;k++)` в) `for(k=1;k<=10;k++)`

```
p=k;
```

```
{ p=k;
```

```
{ p=k; }
```

```
for(j=1;j<=5;j++)
```

```
for(j=1;j<=5;j++)
```

```
for(j=1;j<=5;j++)
```

```
s+= p*j;
```

```
s+= p*j; }
```

```
{ s+= p*j; }
```

28) Якого значення набуде змінна `s` після виконання операторів

```
for(s=0,k=1; k<=3; k++)
```

```
for(j=1; j<=k; j++) s+=j;
```

29) Вказати значення `s` після виконання операторів

```
for(s=0,k=1; k<=2; k++)
```

```
{ m=k;
```

```
do { m++; s+=m; } while( m <= 2);
```

```
}
```

30) Назвати оператори, використовувані для завчасного переривання повторювань циклу.

РОЗДІЛ 5

МАСИВИ В C++

5.1 Поняття масиву

Змінні, якими ми оперували до цих пір, могли зберігати лише одне значення одночасно. За присвоювання їм іншого значення попереднє втрачалося. Часто виникає потреба зберігати велику кількість однотипних значень, які мають опрацьовуватись однаково. Наприклад, екзаменаційні оцінки студентів, які слід вводити, виводити, аналізувати (порівнювати з «2» чи «5»), змінювати за потреби тощо. Такі однотипні значення є сенс зберігати в одній змінній, перенумерувати їх усередині цієї змінної, надати доступ до цих значень за номером і опрацьовувати ці значення у циклі (номер значення збігатиметься з параметром циклу).

Масив у програмуванні – це упорядкована сукупність однотипних елементів. Масиви широко застосовуються для зберігання і опрацювання однорідної інформації, наприклад, таблиць, векторів, матриць, коефіцієнтів рівнянь тощо.

Кожен елемент масиву однозначно можна визначити за ім'ям масиву та індексами. *Ім'я масиву* (ідентифікатор) добирають за тими самими правилами, що й для змінних. *Індекси масиву* визначають місцезнаходження елемента в масиві. Наприклад, елементи вектора мають один індекс – номер по порядку; елементи матриць чи таблиць мають по два індекси: перший означає номер рядка, другий – номер стовпчика. Кількість індексів визначає вимірність масиву. Наприклад, *вектори* у програмах – це одновимірні масиви, *матриці* – двовимірні.

Індексами можуть бути лише змінні, константи чи вирази цілого типу. Значення індексів записують після імені масиву в квадратних дужках. При оголошенні масивів у квадратних дужках зазначається кількість елементів, а нумерація елементів завжди розпочинається з нуля.

Відмінності масиву від звичайних змінних:

- ✓ спільне ім'я для всіх значень;
- ✓ доступ до конкретного значення за його номером (індексом);
- ✓ можливість опрацювання у циклі.

5.2 Одновимірні масиви

5.2.1 Оголошення одновимірних масивів

Одновимірний масив (вектор) оголошується у програмі в такий спосіб:

```
<тип_даних> <ім'я_масиву> [<розмір_масиву>];
```

Тип_даних задає тип елементів масиву. Елементами масиву не можуть бути функції й елементи типу `void`. *Розмір_масиву* у квадратних дужках задає кількість елементів масиву. На відміну від інших мов, у C++ не перевіряється вихід за межі масиву, тому, щоб уникнути помилок у програмі, слід стежити за роз-

мірністю оголошених масивів. Значення *розмір_масиву* при оголошенні масиву може бути не вказано в таких випадках:

- ✓ при оголошенні масив ініціалізується;
- ✓ масив оголошено як формальний параметр функції (докладніше див. розд. 8);
- ✓ масив оголошено як посилання на масив, явно визначений в іншому модулі.

Використовуючи ім'я масиву та індекс, можна звертатися до елементів масиву:

<ім'я_масиву> [<значення_індексу>]

Значення індексів мають перебувати в діапазоні від нуля до величини, на одиницю меншу за розмір масиву, визначений при його оголошенні, оскільки в C++ нумерація індексів розпочинається з нуля.

Наприклад,

```
int A[10];
```

оголошує масив з ім'ям A, який містить 10 цілих чисел; при цьому виділяє і закріплює за цим масивом оперативну пам'ять для усіх 10-ти елементів відповідного типу (*int* – 4 байти), тобто 40 байтів, у такий спосіб:

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

Отже, при оголошенні масиву виділяється пам'ять, потрібна для розташування усіх його елементів. Елементи масиву з першого до останнього запам'ятовуються у послідовно зростаючих адресах пам'яті. Поміж елементами масиву в пам'яті проміжків немає. Елементи масиву запам'ятовуються один за одним поелементно.

Зауважте на звертання у програмі до елементів масиву: A[0] – перший елемент, A[1] – другий, A[9] – останній.

Так, для розміщення елементів одновимірного масиву *int* B[5] виділяється по 4 байти під кожен з 5-ти елементів масиву – тобто усього 20 байт:

Елементи	B[0]				B[1]				B[2]				B[3]				B[4]			
Байти	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Для отримання доступу до *i*-го елемента масиву B, можна написати B[i], де *i* – змінна циклу, яка може набувати значення від 0 до 3. При цьому величина *i* перемножується на розмір типу *int* і являє собою адресу *i*-го елемента масиву B від його початку, після чого здійснюється вибір елемента масиву B за сформованою адресою.

Змінна (ідентифікатор), оголошена як масив, по суті є вказівником на перший елемент масиву (див. розд. 6). Слід зауважити, що у перебігу виконання програми адресу масиву не може бути змінено, хоча значення його елементів можуть змінюватись.

Ось кілька прикладів оголошення масивів:

```
char N[20];
int grades[125];
```

```
float mass[30];
double v[1500];
```

Перший з масивів `N` містить 20 символів. Звертання до елементів масиву може бути таким: `N[0]`, `N[1]`, ..., `N[19]`.

Другий масив `grades` містить 125 цілих чисел. Звертання до елементів такого масиву може бути таким: `grades[0]`, `grades[1]`, ..., `grades[124]`.

Третій масив `mass` містить 30 дійсних чисел. Звертання до елементів масиву може бути таким: `mass[0]`, `mass[1]`, ..., `mass[29]`.

Четвертий масив `v` містить 1500 дійсних чисел з подвійною точністю. Звертання до елементів такого масиву може бути таким: `v[0]`, `v[1]`, ..., `v[1499]`.

При оголошенні масивів можна елементам масиву (необов'язково всім) присвоювати початкові значення, які у подальшому в програмі може бути змінено. Якщо реальна кількість ініціалізованих значень є менше за розмірність масиву, то решта елементів масиву набуває значення 0.

Наприклад,

```
int a[5]={9,33,-23,8,1}; //a[0]=9, a[1]=33, a[2]=-23, a[3]=8, a[4]=1
float b[10] = {1.5, -3.8, 10};
// b[0]=1.5, b[1]=-3.8, b[2]=10, b[3]=b[4]=...=b[9]=0
char s[5]="Фотот"; // s[0]='Ф', s[1]='о', s[2]='т', s[3]='о', s[4]='\0'
char code[]="abc"; //code[0]='a', code[1]='b', code[2]='c', code[3]='\0'
```

В останньому прикладі ініціалізується `code` як масив символів з чотирьох елементів. Четвертим елементом є символ `'\0'`, який завершує усі рядки символів. Якщо рядок є коротше за оголошений розмір масиву символів, то решта елементів масиву ініціалізується нулем (символом `'\0'`).

Масиви в програмах можна оголошувати також зі *створенням типу користувача* (докладніше див. розд. 11):

```
typedef <тип_даних> <ім'я_типу> [<розмір_масиву>];
<ім'я_типу> <ім'я_масиву>;
```

Наприклад, створимо тип з ім'ям `mass` як масив з 10-ти додатних цілих чисел і оголосимо два масиви – `a` та `b` – створеного типу:

```
typedef unsigned short mass[10];
mass a, b;
```

Для оголошування масивів можна використовувати *типізовані констант-масиви*, які дозволяють водночас і оголосити масив, і задати його значення як константи:

```
const <тип_даних> <ім'я_масиву> [<розмір_масиву>] =
    {<значення_елементів_масиву>;};
```

Слід пам'ятати, що змінювати значення елементів констант-масивів не припустимо.

Приклади створювання констант-масивів:

```
const int arr[5]={9,3,-7,0,123}; //масив з 5-ти цілих чисел
const float f[5]={1.5,6,8,7.4,-1.125}; //масив з 5-ти дійсних чисел
const C[5]={15,-6,546}; //масив з 5-ти цілих чисел типу int,
// де C[0]=15, C[1]= -6, C[2]=546, C[3]=0 і C[4]=0
```


5.2.2 Введення-виведення одновимірних масивів

Введення-виведення масиву у консолі

Введення масиву за допомогою cin:

```
for (int i=0; i<n; i++) cin >> a[i];
```

При цьому числа при введенні можуть бути написані як у стовпчик, так і у рядок через пробіл.

Виведення масиву у консолі за допомогою cout у стовпчик:

```
for (int i=0; i<n; i++) cout << a[i] << endl;
```

у рядок через пробіл:

```
for (int i=0; i<n; i++) cout << a[i] << " ";  
cout << endl;
```

Для введення-виведення символьних масивів у консолі використовуються функції `gets()` і `puts()`:

```
char s[50];  
gets(s); // Введення символьного масиву s  
puts(s); // Виведення символьного масиву s
```

Введення елементів одновимірних масивів з різних елементів керування на формі

Вводити значення одновимірних масивів можна на форму, використовуючи різноманітні елементи керування Visual C++ платформи .NET Framework. При цьому вводити значення елементів масивів можна лише поелементно, для чого слід організовувати цикли зі змінням значень індексу зазначеного масиву. Тут розглянемо організацію виведення одновимірних масивів на форму за допомогою елементів керування `richTextBox` та `listBox`.

У подальших прикладах буде використано такий масив: `double A[10];`

За допомогою елемента керування `richTextBox` можна вводити масиви як при виконуванні програми, так і при конструюванні форми проекту програми через вікно властивості `Lines` (для переходу до нового рядка при введенні значень треба натиснути клавішу [Enter]).

Приклад фрагмента програми введення значень елементів одновимірного масиву `A` (в кожному рядку по одному числу):

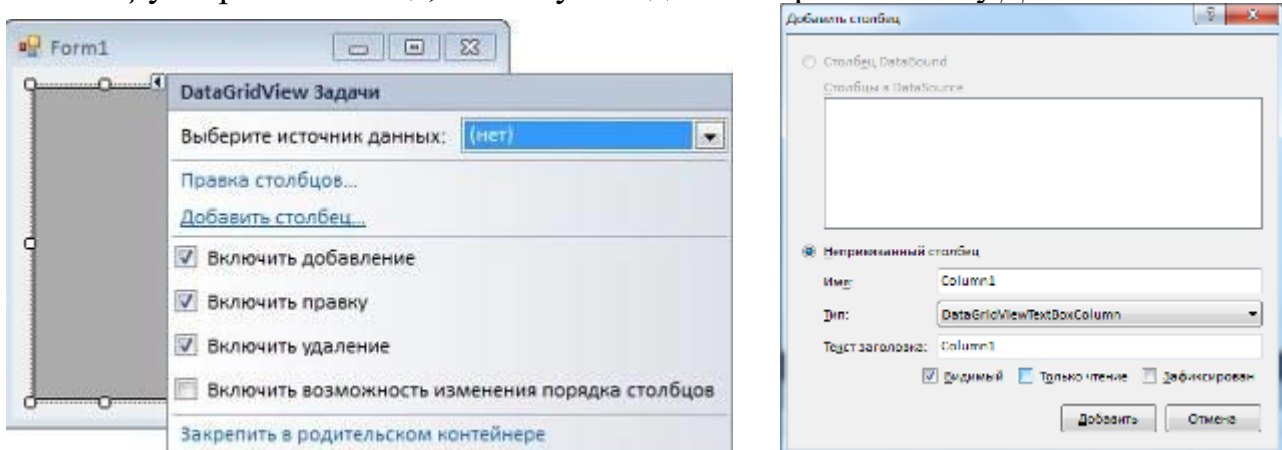
```
for (i=0;i<=9;i++) A[i]= Convert::ToDouble(richTextBox1->Lines[i]);
```

За допомогою елемента керування `listBox` можна вводити масиви так само, як і з елементом керування `richTextBox`, лише замість властивості `Lines` використовувати властивість `Items`.

```
for (i=0;i<=9;i++) A[i]= Convert::ToDouble(listBox1->Items[i]);
```

Елемент керування `dataGridView`, який також можна використовувати для введення-виведення одновимірних масивів, має вигляд таблиці з клітинками, в кожній з яких можна розмістити величину рядкового типу (`String^`). Щоб звернутись до значення `Value` конкретної клітинки слід зазначити номер рядка `Rows[i]` і номер стовпця `Cells[j]`: `dataGridView1->Rows[i]->Cells[j]->Value`. або `dataGridView1[j,i]->Value`.

Після розміщення на формі елемента `dataGridView` через діалогове вікно *DataGridView Задачі*, яке автоматично з'явиться, слід командою *Добавить столбец* утворити стовпц, натиснувши декілька разів кнопку *Добавить*.



За допомогою команди *Правка столбцов* діалогового вікна *DataGridView Задачі* можна налаштувати параметри кожного зі стовпців, наприклад, зменшити ширину (`Width`) стовпців від 100 за промовчанням до 50 пікселів.

Створювати (додавати) нові рядки можна лише у програмному коді, наприклад, створивши функцію `Form1_Load` і вписавши в неї команду створення десяти рядків:

```
dataGridView1->Rows->Add(10);
```

Також можна задати значення `False` для таких властивостей `DataGridView`:

- `ColumnHeadersVisible` (заголовок стовпця);
- `RowHeadersVisible` (стовпець із заголовками рядків);
- `AllowUserToAddRows` (дозвіл на автоматичне долучення рядків).

Докладніше використання цього елемента буде розглянуто у прикладах 5.4 і 5.7 та підрозд. 5.3.

Виведення елементів одновимірних масивів до різних елементів керування

Виведення до багаторядкового елемента керування `richTextBox` елементів одновимірних масивів можна організовувати у стовпчик, відокремлюючи елементи символом `"\n"`. Приклад фрагмента програми виведення масиву `A`:

```
richTextBox1->Clear(); // Очищення елемента керування
// Почергове виведення елементів масиву у циклі
for (i=0; i<=9; i++)
    richTextBox1->Text+= A[i].ToString()+"\n";
```

Останню команду можна замінити на:

```
richTextBox1->AppendText(x.ToString()+"\n");
```

Виведення масивів до елемента керування `listBox` організовують у властивість `Items`, яка є контейнером елементів списку:

```
listBox1->Items->Clear();
for (i=0; i<=9; i++) listBox1->Items->Add(x.ToString()+"\n");
```

Ще раз звернімося до питання стосовно доцільності використання у програмах масивів замість окремих змінних. Порівняймо застосування масиву з 7-ми цілих чисел і 7-ми окремими змінними:

Масив з 7-ми цілих чисел	7 окремих цілих змінних
Оголошення	
<code>int a[7];</code>	<code>int a, b, c, d, e, f, g;</code>
Введення	
<code>for(int i=0; i<n; i++) cin>>a[i];</code>	<code>cin>>a; cin>>b; cin>>c; cin>>d; cin>>e; cin>>f; cin>>g;</code>
Обчислення суми додатних елементів	
<code>double s=0; for (int i=0; i<7; i++) if (a[i]>0) s+=a[i];</code>	<code>if (a>0) s=a; if (b>0) s+=b; if (c>0) s+=c; if (d>0) s+=d; if (e>0) s+=e; if (f>0) s+=f; if (g>0) s+=g;</code>

На практиці звичайно у масивах зберігається набагато більше 7-ми елементів і використання окремих змінних стає, м'яко кажучи, недоцільним.

5.2.3 Програмування базових алгоритмів опрацювання одновимірних масивів

Опрацювання масиву полягає у виконанні операцій над його елементами. Окрім введення-виведення масивів існує перелік найпоширеніших базових алгоритмів опрацювання масивів:

- ✓ обчислення узагальнених характеристик (сум, добутків і кількості елементів) (див. приклади 5.1 ... 5.3);
- ✓ пошук максимального чи мінімального елемента (див. приклади 5.4, 5.5);
- ✓ пошук заданих елементів (див. приклад 5.2 ... 5.9);
- ✓ переставляння елементів (див. приклади 5.6, 5.9);
- ✓ упорядкування (сортування) масивів (див. приклад 5.14).

Базові алгоритми опрацювання масивів можуть бути складовими блоками при розв'язуванні більш складних задач. А тому організовувати їх доцільно у вигляді окремих функцій (див. п. 5.2.4 та розд. 8).

Приклад 5.1. Ввести одновимірний масив з 8-ми дійсних чисел та обчислити суму всіх елементів масиву.

Текст консольної програми та блок-схема:

```
#include <iostream>
using namespace std;
int main()
{ setlocale(0, ".1251");
```

```
double A[8], sum = 0;
cout << "Введіть 8 чисел" << endl;
for (int i=0; i<8; i++)
{ cin >> A[i];    // введення елементів масиву
  sum += A[i];    // обчислення суми елементів
}
cout << "Сума елементів масиву = " << sum << endl;
system ("pause>>void");
return 0;
}
```

Результати роботи:

Елементи масиву можна вводити через *Enter*:

Введіть 8 чисел:

0.1
-72
3.5
12.7
50
4.6
2
0.2

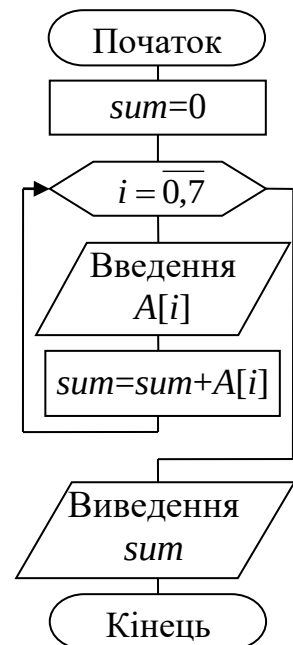
Сума елементів масиву = 1.1

Числа також можна ввести через пробіл:

Введіть 8 чисел:

1 2 3 4 5 6 7 8

Сума елементів масиву = 36



Приклад 5.2. Скласти консольну програму, яка вводить масив з 12-ти цілих чисел й обчислює кількість непарних елементів з проміжку (−10, 30).

Текст програми в онлайн-компіляторі:

```
#include <iostream>
using namespace std;
int main()
{ int a[12], i, k=0;
  for (i=0; i<12; i++)
  { cin >> a[i];
    if ((a[i]%2 != 0) && (a[i] > -10) && (a[i] < 30)) k++;
  }
  cout << "k=" << k << endl;
}
```

Результати виконання програми:

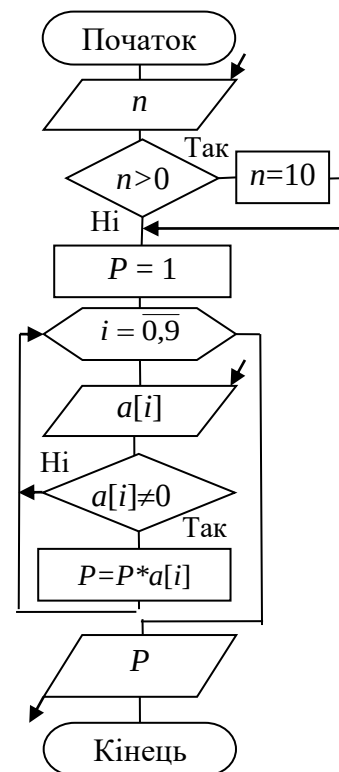
1 3 0 4 55 7 8 -30 -2 510 1 22
k=4

Приклад 5.3. Ввести за допомогою форми масив до 10-ти цілих чисел та обчислити добуток ненульових елементів масиву.

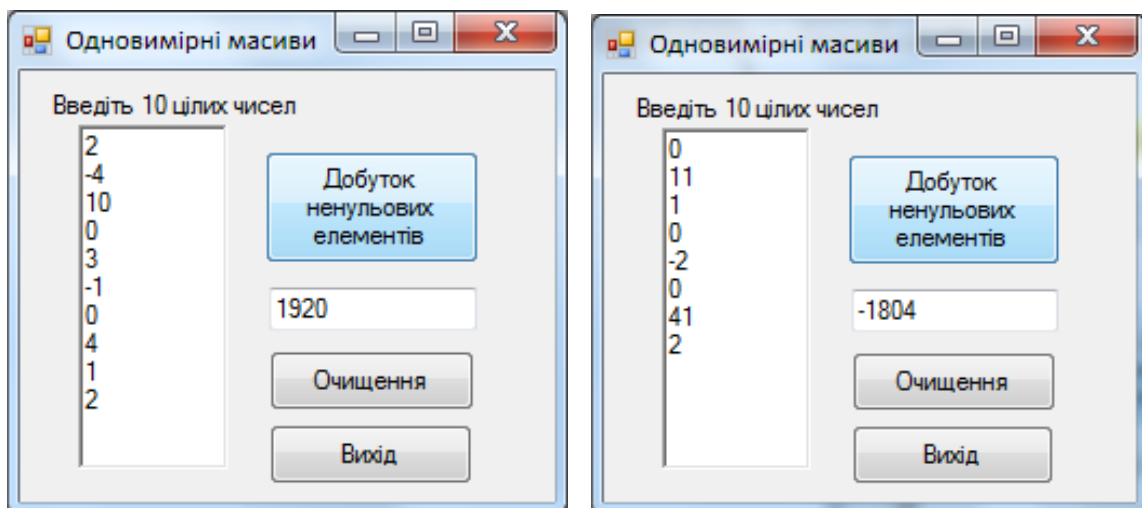
Розв'язок. Вводити масиви можна у елемент `richTextBox`.

Текст програми та блок-схема:

```
private: System::Void button1_Click(System::Object^
                                     sender, System::EventArgs^ e)
{ int A[10], p=1, n=richTextBox1->Lines->Length;
  if (n>10) n=10;
  for (int i=0; i<n; i++)
  { A[i]=Convert::ToInt32(richTextBox1->Lines[i]);
    if (A[i]!=0) p*=A[i];
  } textBox1->Text=Convert::ToString(p);
}
private: System::Void button2_Click(System::Object^
                                     sender, System::EventArgs^ e)
{ richTextBox1->Clear(); textBox1->Clear();
}
private: System::Void button3_Click(System::Object^
                                     sender, System::EventArgs^ e)
{ Close(); }
```



Результати роботи:



Приклад 5.4. Ввести масив із 9-ти дійсних чисел, обчислити найменший елемент масиву та його індекс.

Розв'язок. Розглянемо алгоритм пошуку мінімального елемента масиву. Припустимо, що на столі лежить купа резюме, кожне з яких, окрім іншої інформації про кандидата на посаду, містить і його рік народження. Треба віднайти найстаршого з усіх кандидатів, тобто резюме з найменшим роком народження.

Беремо до рук перше резюме. Оскільки невідомо, що міститься у решті резюме, відкладаємо перше і вважаємо його власника за найстаршого.

Одне за одним переглядаємо решту резюме. Якщо натрапляємо на резюме, в якому подано рік менше за рік у відкладеному резюме, то відкладаємо його (попереднє нас вже не цікавить). Коли всі резюме переглянуто, виявляється, що відкладене резюме належить кандидатові з найменшим роком народження.

У вигляді словесного алгоритму пошук мінімального елемента масиву і його індексу можна записати в такий спосіб:

1) вважаємо перше число за найменше. Для цього присвоюємо перший елемент масиву $A[0]$ у змінну min , а змінній ind – значення 0 (тобто індекс першого елемента масиву);

2) у циклі переглядаємо всі елементи. Якщо зустрічаємо число, яке є менше за min , запам'ятовуємо це число у min , а його індекс – в ind .

Коли цикл завершиться, у змінній min буде зберігатися найменший елемент масиву, а його індекс – у змінній ind . Оскільки нумерація індексів масиву розпочинається з 0, а не з 1, як у повсякденному житті, виветься значення індексу, збільшене на 1.

Крім елемента `richTextBox`, масиви також можна вводити (чи виводити) в елемент керування `dataGridView`. При розміщенні його на формі автоматично сформується діалогове вікно *DataGridView Задачі*, в якому слід вибрати команду *Добавить столбец*. У вікні, що відкриється, слід натиснути кнопку *Добавить*, при цьому сформується один стовпець. (Кожному стовпцю можна задавати власне ім'я та вибирати тип даних. Усталено перший створений стовпець матиме ім'я `Column1` і тип даних, який надає можливість введення-виведення тексту.)

Позаяк для введення одновимірного масиву достатньо одного стовпця, більше стовпців створювати не треба, а просто натиснути кнопку *Закреть*.

Далі слід вибрати значення `False` для трьох властивостей `dataGridView`:

- `ColumnHeadersVisible` (заголовок стовпця);
- `RowHeadersVisible` (стовпець із заголовками рядків);
- `AllowUserToAddRows` (дозвіл на автоматичне долучення рядків).

Крім вже налаштованого елемента `dataGridView` слід на панелі елементів⁵ вибрати і розмістити на формі дві командні кнопки `button`, два `textBox` для виведення результатів та три `label` для створення пояснювальних підписів.

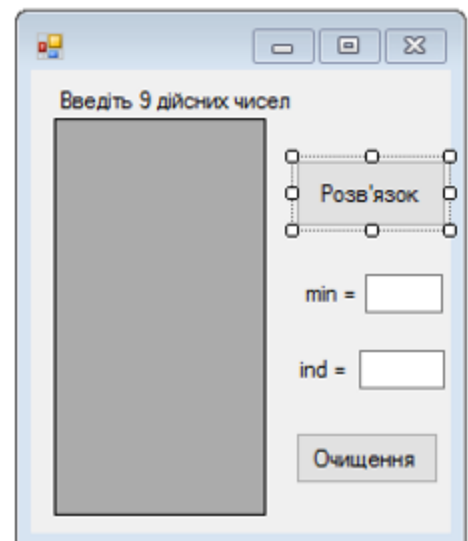
Після остаточних налаштувань форма набуде вигляду →

Перед написанням програмного коду слід зазначити ще одну особливість `dataGridView`: рядки можна долучати лише у програмі. Саме тому слід подвійним клацанням миші по формі створити функцію `Form1_Load` та вписати в неї команду створення 9-ти рядків:

```
dataGridView1->Rows->Add(9);
```

Текст програми та блок-схема:

```
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
    dataGridView1->Rows->Add(9);
}
```



⁵ Якщо в даний момент Ви не бачите панель елементів, її можна показати командою меню *Вид / Панель елементів (View/Toolbox)*, або клавішами `Ctrl+Alt+x`.

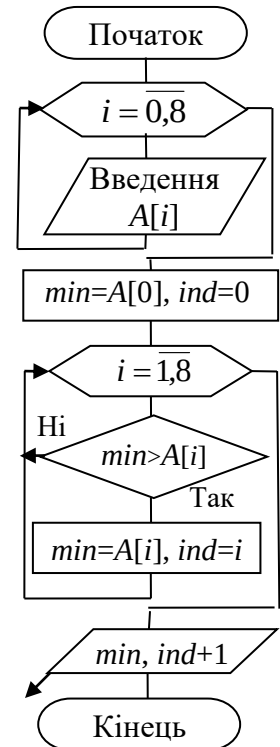
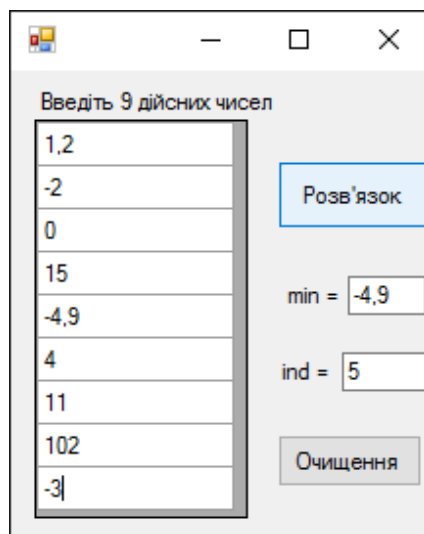
```

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{ double a[9], min; int i, ind;
  for(i=0; i<9; i++)
    a[i]=Convert::ToDouble(dataGridView1->Rows[i]->Cells[0]->Value);
  min=a[0]; ind=0;
  for (i=1; i<9; i++)
    if (a[i]<min) { min=a[i]; ind=i;}
  textBox1->Text = min.ToString();
  textBox2->Text = (ind+1).ToString();
}

private: System::Void button2_Click(System::Object^ sender,
System::EventArgs^ e)
{ for (int i=0; i<9; i++)
  dataGridView1->Rows[i]->Cells[0]->Value = "";
  textBox1->Clear();
  textBox2->Clear();
}

```

Результати роботи:



Приклад 5.5. Обчислити 10 елементів масиву за формулою $a_i = (-1)^i \frac{\sin(i^2)}{\sin(i+1)}$

та визначити найбільший від'ємний елемент масиву.

Розв'язок. Оскільки невідомо, чи буде перший елемент масиву від'ємним, не можна присвоїти початкове значення змінній *max*. Присвоїмо їй значення 0. Вважатимемо, що, якщо зустріли від'ємний елемент і змінна *max* дорівнює 0, то це буде перший від'ємний елемент і його слід присвоїти змінній *max* у якості початкового значення. Якщо змінна *max* вже набула певного значення, порівнюємо її з новим від'ємним елементом, і якщо новий елемент перевищує *max*, записуємо його до *max*.

Текст програми в онлайн-компіляторі:

```

#include <iostream>
#include <math.h>
int main()
{ double a[10], max=0; int i;
  for(i=0; i<10; i++)
  { a[i]=pow(-1,i)*sin(i*i)/sin(i+1); // Обчислення елемента вектора
    printf("%5.2f ", a[i]);           // Виведення елемента вектора
  }
}

```



```

for (i=0; i<10; i++)
    if (a[i]<0) // Якщо елемент від'ємний i,
        if (max==0 || max<0 && max<a[i]) // якщо він є першим чи більшим за max,
            max=a[i]; // присвоїти його max
printf("\n max = %5.2f", max);
return 0;
}

```

Результати виконання програми:

```

0.00 -0.93 -5.36  0.54  0.30 -0.47 -1.51  0.96  2.23 -1.16
max = -0.47

```

Приклад 5.6. Ввести одновимірний масив до 15-ти елементів та переставити у ньому максимальний елемент з його з першим елементом вектора.

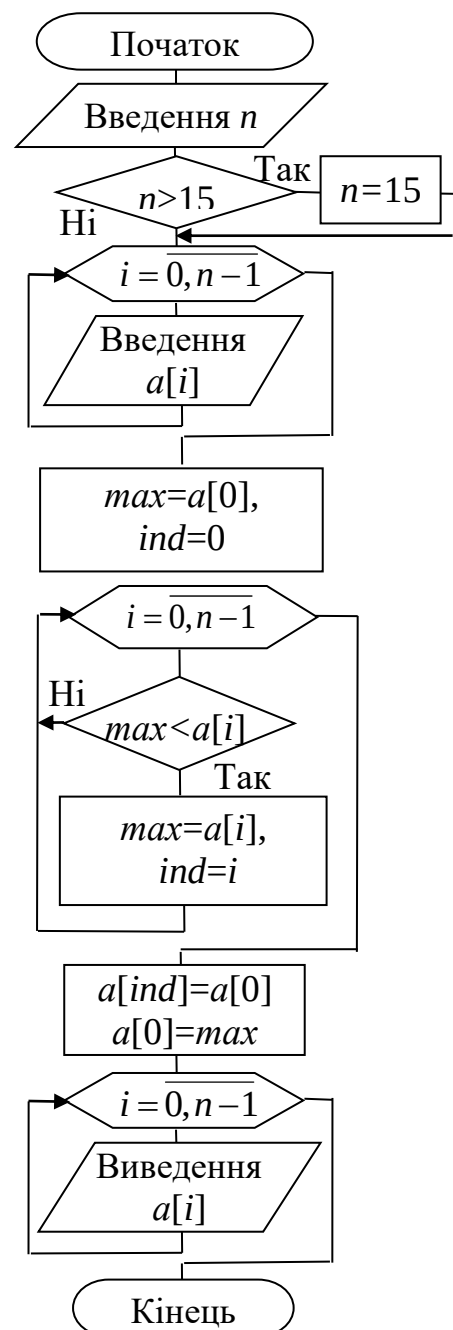
Розв'язок. Зверніть увагу, що чисел може бути введено менш за 15, але оголосити масив слід з максимально можливою кількістю елементів (15). У програмі при опрацюванні масиву індекс у циклі змінюється від 0 до n , де n – кількість реально введених чисел. Якщо введено менше за 15 чисел, зчитуються всі числа, які було введено. Якщо чисел є більше за 15, зайвими числами нехтують.

Текст програми:

```

private: System::Void button1_Click(System::Object^
sender, System::EventArgs^ e)
{
    double a[15], max;
    int i=0, ind=0, n=richTextBox1->Lines->Length;
    if (n>15) n=15;
    // Почергове введення елементів масиву
    for (i=0; i<n; i++)
        a[i]=Convert::ToDouble(richTextBox1->Lines[i]);
    max=A[0];
    for (i=0; i<n; i++)
        if (A[i] > max)
        { max=A[i]; // Максимальний елемент
          ind=i;   // та його індекс
        }
    // Поміняти місцями максимальний елемент
    // з першим елементом вектора
    A[ind]=A[0]; A[0]=max;
    richTextBox2->Clear();
    // Почергове виведення елементів масиву
    for (i=0; i<n; i++)
        richTextBox1->AppendText(a[i].ToString()+"\n");
}

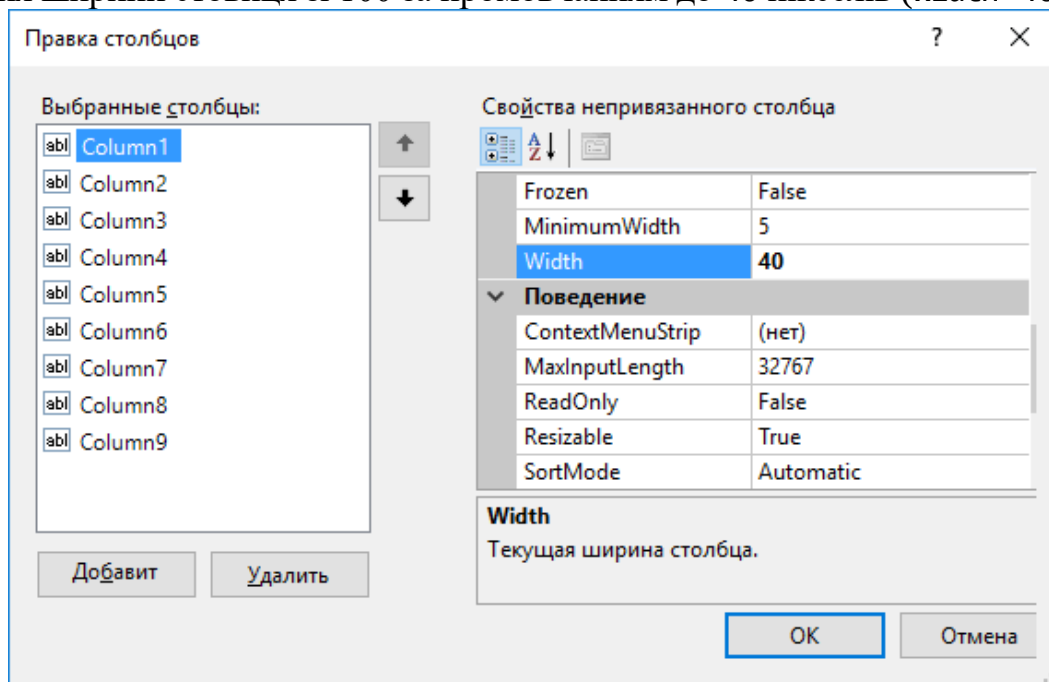
```



Приклад 5.7. Ввести масив з 10-ти цілих чисел. Створити новий масив, елементи якого обчислити, поділивши кожен з елементів початкового масиву на суму його елементів з непарними індексами.

Розв'язок. Щоб створити новий масив, спочатку обчислимо суму елементів з непарними індексами. Після цього послідовно поділимо всі елементи масиву на обчислену суму і присвоїмо результат відповідному елементові нового масиву. Після ділення елементи нового масиву матимуть тип `double`.

Будемо вводити початковий масив і виводити новий масив у елементи керування `dataGridView`. Цей елемент керування зазвичай використовується для таблиць. Він складається з комірок, кожна з яких є розміщена на перетині рядка і стовпця і має два індекси. У цьому прикладі елементи керування `dataGridView` матимуть імена (властивість `Name`) `dGV1` та `dGV2` і міститимуть по одному рядку (створюватимуться програмно при завантаженні форми) і 10 стовпчиків (у властивості `Columns`, який є колекцією властивостей кожного створеного стовпця). Кожному стовпцю можна задавати власне ім'я, вибирати тип даних та задавати його властивості. У цьому прикладі для всіх десяти доданих стовпців доречно змінити значення ширини стовпця зі 100 за промовчанням до 40 пікселів (`width=40`).



Крім того, слід вибрати значення `False` для трьох властивостей `dataGridView`:

- `ColumnHeadersVisible` (заголовок стовпця);
- `RowHeadersVisible` (стовпець із заголовками рядків);
- `AllowUserToAddRows` (дозвіл на автоматичне долучення рядків).

Щоби не виконувати налаштування всіх зазначених властивостей двічі для кожного `dataGridView`, можна налаштувати один, скопіювати його та змінити ім'я другого.

Текст програми:

```
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{ dGV1->Rows->Add(1);
  dGV2->Rows->Add(1);
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    const int N=10;           // Константа N для кількості елементів масиву
    int a[N], i, sum=0;       // a - початковий масив
    double b[N];              // b - новий масив
    for(i=0; i<N; i++)        // Введення елементів початкового масиву
        a[i]=Convert::ToInt32(dGV1[i,0]->Value); // із StringGrid (стовпчик i, рядок 0)
    for(i=0; i<N; i+=2)       // Обчислення суми елементів з непарними індексами
        sum+=a[i];
    textBox1->Text=sum.ToString();
    for(i=0; i<N; i++)
    { b[i]=1.*a[i]/sum;        // Ділимо a[i] на sum і присвоюємо результат b[i]
      dGV2[i,0]->Value=b[i].ToString("0.00");
    }
}
```

Приклад 5.8. Ввести масив до 10-ти дійсних чисел і створити новий масив з його ненульових елементів.

Розв'язок. Оскільки реальна кількість ненульових елементів початкового масиву є невідома, оголошуємо новий масив з максимально можливою кількістю елементів, тобто 10. У попередньому прикладі індекси елементів нового масиву при створенні збігалися з індексами відповідних елементів початкового масиву. У цьому прикладі нульові елементи пропускаються і не записуються до нового масиву, тому індекси початкового й результуючого масивів не збігатимуться.

Конструювання форми з двома елементами `dataGridView` є подібним до попереднього прикладу 5.7.

Текст програми:

```
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
    dGV1->Rows->Add(1);          dGV2->Rows->Add(1);
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    const int N=10;
    double a[N],b[N]={0}; // Спочатку елементи нового масиву мають нульові значення
    int i,j=0;             // Кількість елементів нового масиву дорівнює 0
    for (i=0; i<N; i++)
    { a[i] = Convert::ToDouble(dGV1[i,0]->Value);
      if (a[i]!=0)          // Якщо елемент a[i] є ненульовим,
      { b[j]=a[i]; j++;     // записати його до b[j] та збільшити j
      }
    }
    dGV2->ColumnCount=j; // Тепер вже є відома реальна кількість комірок SG2
    for (i=0; i<j; i++) dGV2[i,0]->Value = b[i].ToString();
}
```

Приклад 5.9. Ввести масив з 8-ми дійсних чисел і видалити з нього всі від'ємні елементи.

Текст консольної програми:

```
#include <iostream>
using namespace std;
int main()
{ setlocale(0, ".1251");
  double a[8]; int i,j,n=8;
  cout<<"Введіть масив з 8-ми дійсних чисел:\n ";
  for (i=0; i<n; i++) cin>>a[i]; // Введення масиву
  for (i=0; i<n; i++)
  if (a[i]<0)
  { for (j=i; j<n; j++) a[j]=a[j+1]; // Видалення елемента з індексом j
    n--; i--;
  }
  cout<<"\nМасив без від'ємних з "<<n<<" елементів: \n ";
  for (i=0; i<n; i++) cout<<a[i]<<"\t"; // Виведення масиву
  system("pause>>void");
  return 0;
}
```

Результати роботи:

Введіть масив з 8-ми дійсних чисел:
-4 3.5 0 -2.3 39 -10 45 9

Масив без від'ємних з 5 елементів:
3.5 0 39 45 9

5.2.4 Опрацювання одновимірних масивів у функціях

Більш докладно правила організації функцій розглядаються у розд. 8. Тут лише зауважимо, що *функцією* є окремо організований поіменований блок команд програми, в якому розв'язується невелика і специфічна частина загального завдання. Організація програмного проекту з розподілом великого складного завдання на набір більш простих задач і завдань, розв'язування яких виокремлено в логічні блоки – функції, дозволяє спростити читаність програми, її розуміння. Окрім того, застосування функцій вивільняє програмний код від повторюваних блоків команд, які багаторазово реалізують один і той самий алгоритм для різних даних. У таких випадках створюється функція з певним набором операторів, а виклик цієї функції здійснюють в основній програмі потрібну кількість разів для різних вхідних параметрів.

Якщо результатом виконання функції є одне значення, яке повертається оператором `return`, то воно підставляється у точку виклику функції. Тому тип функції і є типом результату. Коли функція має повертати масив, типом функції слід оголосити вказівник на тип елемента масиву. Наприклад, оголошена в такий спосіб функція може повертати масив цілих чисел:

```
int * function ();
```

При опрацюванні масивів у функціях передавання їх у якості аргументів завжди здійснюється за адресою, тобто передається адреса першого елемента (початок масиву), а доступ до кожного з елементів масиву здійснюється як певний зсув (обчислюваний через індекси) від початку масиву.

Передавання одновимірних масивів до функцій у якості аргументів можна організувати одним з трьох способів:

```
double fun (int a[10]);  
double fun (int a[]);  
double fun (int *a);
```

За першим способом явно зазначено кількість елементів масиву. У другій синтаксичній формі константний вираз у квадратних дужках є відсутній. За третього способу передається вказівник як посилання на масив, явно визначений в основній програмі. Більш докладно роботу з вказівниками буде розглянуто в розд. 6.

Оскільки для другого і третього способів інформація про кількість елементів у прототипі відсутня, то такі форми припустимі, коли кількість елементів масиву є глобальними змінними чи константами. Але доцільніше передавати розмірність одновимірного масиву до функції окремим параметром, позаяк це зробить таку функцію більш універсальною, адже вона зможе коректно опрацювати масиви з різною кількістю елементів.

```
double fun (int a[], int n);  
double fun (int *a, int n);
```

Хоча наведені записи мають різний синтаксис, насправді вони є рівнозначними, позаяк авторами стандарту C для більшої ясності було вирішено, що ма-

сив оголошений як параметр функції є **вказівником**. При цьому навіть явно зазначене число у квадратних дужках ігнорується.

Якщо функцію організовано для опрацювання елементів масиву, наприклад, для сортування елементів, у такому разі можна оголосити функцію з типом результату **void** (немає величини, що повертається). Позаяк сам масив передається у функцію за посиланням, то будь-які змінювання значень елементів масиву у функції буде видно і в основній програмі, яка викликає цю функцію (див. нижче приклади 5.12, 5.14 ... 5.16).

Приклад 5.10. Створити функцію обчислення середнього арифметичного значення ненульових елементів цілочисельного масиву та за допомогою цієї функції визначити середнє арифметичне ненульових елементів для введеного масиву з 10-ти цілих чисел: спочатку для всіх елементів, а тоді для першої половини масиву.

Схеми алгоритму функції та основної програми:

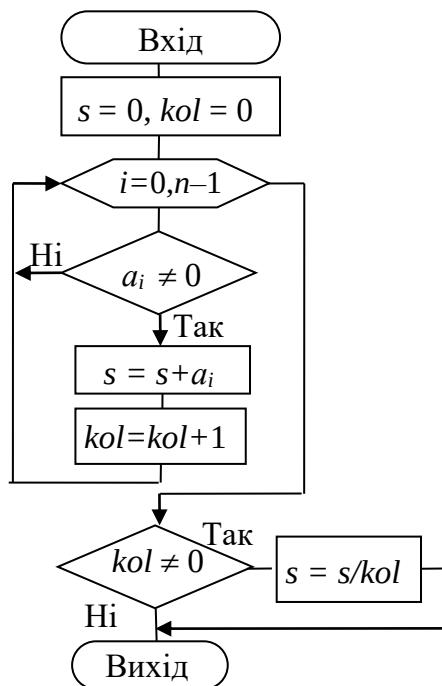


Схема алгоритму
функції **sr()**

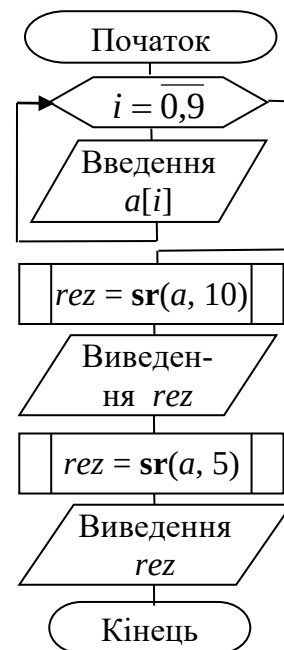


Схема алгоритму
до основної програми

Текст програмного коду та схема алгоритму:

```

#include <iostream>
using namespace std;
double sr(int a[], int n)
{ double s=0; int i, kol=0;
  for (i=0; i<n; i++)
    if (a[i]) { s+=a[i]; kol++; }
  if (kol) s/=kol;
  return s;
}
  
```

```

int main ()
{ setlocale(0, ".1251");
  int a[10], i;
  cout<<" Введіть 10 цілих чисел:\n";
  for(i=0; i<10; i++) cin >> a[i];
  cout<<"\n Середнє арифметичне "<<10<<" елементів - " <<sr(a,10) <<endl;
  cout<<" Середнє арифметичне "<< 5 <<" елементів - " << sr(a,5) <<endl;
  system ("pause>>void");
  return 0;
}

```

Результати роботи консольної програми:

Введіть 10 цілих чисел:

6 -8 0 3 1 34 -5 1 0 2

Середнє арифметичне 10 елементів - 4.25

Середнє арифметичне 5 елементів - 0.5

Приклад 5.11. Створити функцію визначення максимального і мінімального елементів масиву з 10-ти дійсних чисел та організувати її виклик для введеного масиву.

Схеми алгоритму функції та основної програми:

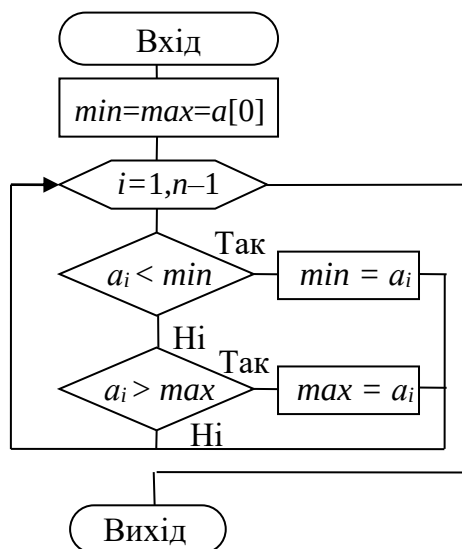


Схема алгоритму
функції **MinMax ()**

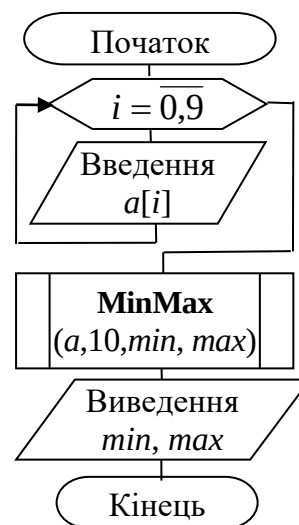


Схема алгоритму
до основної програми

Текст функції та основної програми:

```

#include <iostream>
using namespace std;

void MinMax (double a[], int n, double& min, double& max)
{ min=max=a[0];
  for (int i=1; i<n; i++)
  { if (a[i]<min) min=a[i];
    if (a[i]>max) max=a[i];
  }
}

```

```

int main ()
{ setlocale(0, ".1251");
  double a[10], min, max;
  cout<<"Введіть 10 чисел:\n";
  for (int i=0; i<10; i++) cin >> a[i];
  MinMax(a, 10, min, max);
  cout<<" Мінімальне значення: "<< min << endl;
  cout<<" Максимальне значення: "<< max << endl;
  system ("pause>>void");
  return 0;
}

```

Результати роботи консольної програми:

Введіть 10 чисел:

1.2 0 -3.8 12 3 34.5 7 0 -54 14

Мінімальне значення: -54

Максимальне значення: 34.5

Приклад 5.12. Створити функцію для обміну місцями максимального та мінімального елементів масиву і перевірити правильність роботи функції для масиву розмірністю до 12-ти цілих чисел.

Текст функції та основної програми:

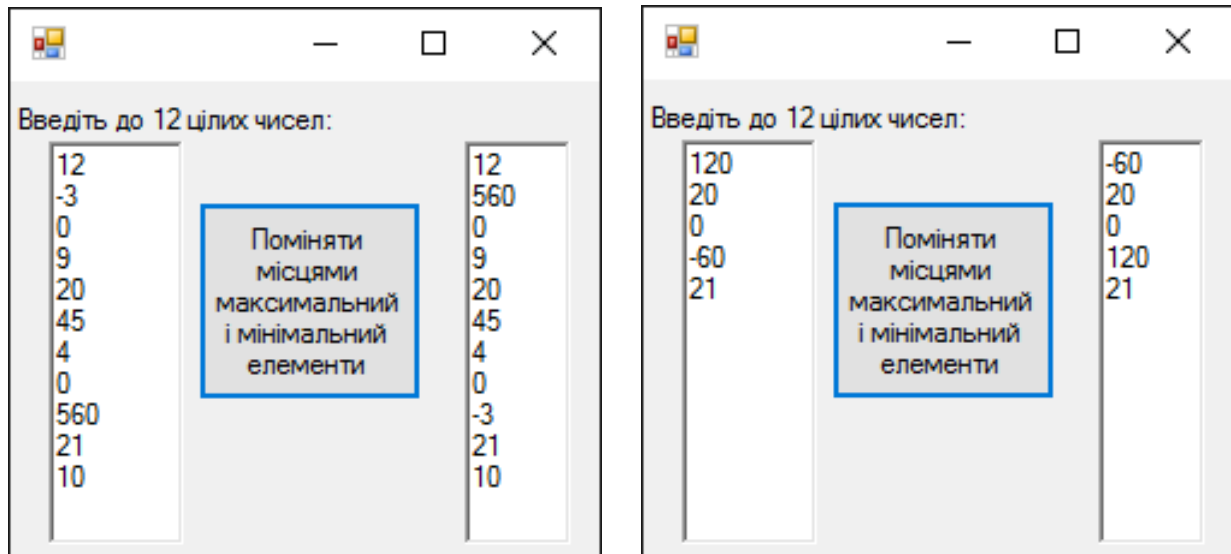
```

void change(int a[], int n)
{
  int imin=0, imax=0, tmp;
  for (int i=0; i<n; i++)
  {
    if(a[imin]>a[i]) imin=i;
    if(a[imax]<a[i]) imax=i;
  }
  tmp=a[imin];
  a[imin]=a[imax];
  a[imax]=tmp;
}

private: System::Void button1_Click(System::Object^sender, System::EventArgs^e)
{
  int a[12], i, n=richTextBox1->Lines->Length;
  if (n>12) n=12;
  for (i=0; i<n; i++) a[i]=Convert::ToInt32(richTextBox1->Lines[i]);
  change(a, n);
  richTextBox2->Clear();
  for (i=0; i<n; i++) richTextBox2->Text+=Convert::ToString(a[i])+"\n";
}

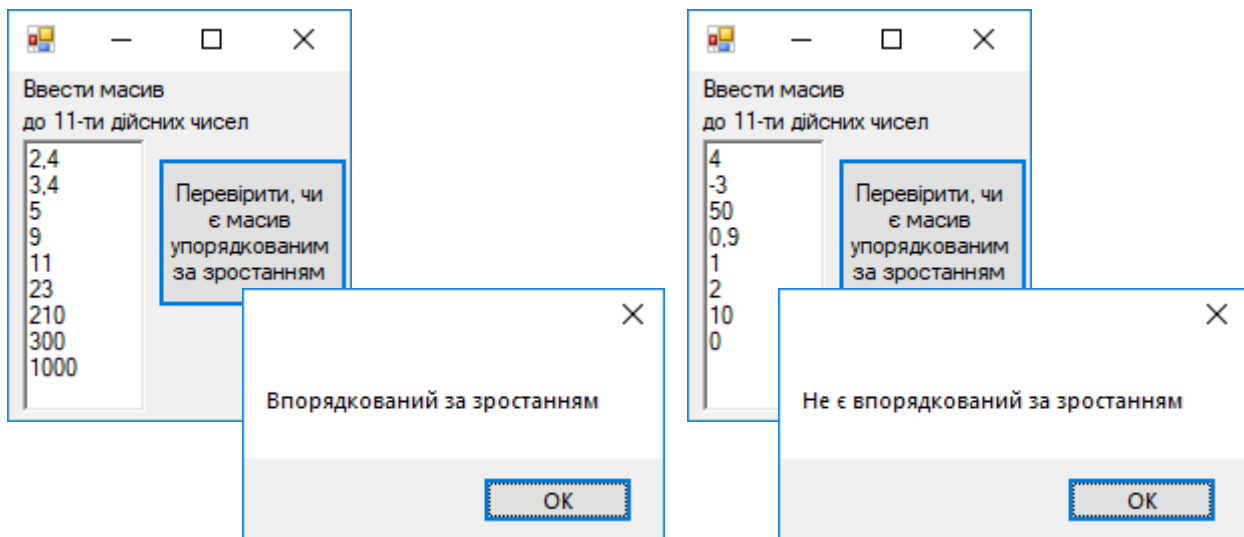
```

Результати роботи:



Приклад 5.13 Ввести масив до 11-ти дійсних чисел і з'ясувати, чи є він упорядкованим за зростанням.

Розв'язок 1. У циклі порівнюватимемо кожен парі сусідніх елементів. Якщо принаймні для однієї пари виконається умова ($a[i] > a[i+1]$) (наступний елемент є менший за попередній), це означатиме, що масив не є впорядкований, і в такому разі слід вивести про це повідомлення і перервати виконання функції `button1_Click`. Якщо цикл завершиться, а виконання функції перервано не було, це означатиме, що масив є впорядкований, про що виведеться відповідне повідомлення.



```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    double a[11];
    int i, N=richTextBox1->Lines->Length;
    if (N>11) N=11;
    for (i=0; i<N; i++) a[i]=Convert::ToDouble(richTextBox1->Lines[i]);
    for (i=0; i<N-1; i++)
```

```

    if (a[i] > a[i+1])
    { MessageBox::Show("Не є впорядкований за зростанням"); return;}
    MessageBox::Show("Впорядкований за зростанням");
}

```

Розв'язок 2. Напишемо аналогічну програму зі створенням функції `sorted()`, яка повертатиме значення `true`, якщо масив є впорядкований за зростанням, і `false` – якщо ні. У циклі порівнюватимемо кожну пару елементів, і, якщо зустрінеться пара, для якої наступний елемент є меншим за попередній, перервемо виконання функції й повернемо `false`. Якщо цикл завершено і виконання функції перервано не було, повернемо `true`. Докладніше про специфіку організації функцій див. у розд. 8.

```

bool sorted(double a[], int n)
{ for (int i=0; i<n-1; i++)
    if (a[i] > a[i+1]) return false;
  return true;
}

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{ double a[11];
  int i, N=richTextBox1->Lines->Length;
  if (N>11) N=11;
  for (i=0; i<N; i++) a[i]=Convert::ToDouble(richTextBox1->Lines[i]);
  if (sorted(a, N)) MessageBox::Show("Впорядкований за зростанням");
  else MessageBox::Show("Не є впорядкований за зростанням");
}

```

Приклад 5.14. Ввести масив із 12-ти дійсних чисел і впорядкувати (відсортувати) його за зростанням значень.

Розв'язок. Існують різні методи сортування масиву. Вони розрізняються переважно швидкістю отримання результату. Розглянемо лише три з них.

Перший спосіб

У наведеній нижче функції `sort()` запропоновано найбільш поширений метод «бульбашки» (bubblesort). Алгоритм сортування полягає у переставлянні двох сусідніх порівнюваних елементів масиву, якщо вони йдуть у неправильному порядку.

Перш ніж розглянемо повний текст програм, звернімо увагу на певні її фрагменти.

Для послідовного порівняння сусідніх елементів масиву `a[i]` та `a[i+1]` слід організувати цикл. При цьому останнє значення параметра циклу `i` у циклі `for` має бути на один менше за кількість усіх елементів, щоб на останній ітерації можна було порівнювати передостанній елемент з останнім.

Для того щоб переставити місцями два елементи, слід застосувати тимчасову змінну `temp`:

Для переставляння двох елементів застосовується тимчасова змінна `temp`:

```

temp = a[i];    // Запам'ятати значення a[i],
a[i] = a[j];    // в a[i] записати a[j],
a[j] = temp;    // а в a[j] записати те, що спочатку було в a[i]

```


Щоб краще зрозуміти необхідність тимчасової змінної, слід уявити собі склянку соку і чашку кави і спробувати перелити сік у чашку, а каву – у склянку. Це неможливо буде зробити без додаткового посуду, до якого спочатку треба перелити вміст, наприклад, склянки.

Якщо в умові порівняння елементів записати знак «>», то елементи сортуватимуться за зростанням, а якщо знак «<» – за спаданням.

Цикл для одноразового проходження по n елементах масиву і за потреби переставлення елементів має вигляд

```
for (i=0; i<n-1; i++)
    if (a[i] > a[i+1])
    { temp = a[i];
      a[i] = a[i+1];
      a[i+1] = temp;
    }
```

Перевіримо роботу цього циклу на масиві з 7-ми елементів:

5, -1.3, -7.1, 4.8, 2, 7, 1.6.

Після виконання наведеного вище циклу цей масив набуде вигляду

-1.3, -7.1, 4.8, 2, 5, 1.6, 7

тобто масив ще не є остаточно відсортованим за зростанням і дію наведеного вище циклу слід повторювати, допоки масив не набуде вигляду

-7.1, -1.3, 1.6, 2, 4.8, 5, 7.

У поданій нижче таблиці наведено результати чотирьох проходів масивом. Після останнього проходження масив вже відсортовано.

	Елементи масиву						
	0	1	2	3	4	5	6
початок	5	-1.3	-7.1	4.8	2	7	1.6
1-й крок	-1.3	7.1	4.8	2	5	1.6	7
2-й крок	-7.1	-1.3	2	4.8	1.6	5	7
3-й крок	-7.1	-1.3	2	1.6	4.8	5	7
4-й крок	-7.1	-1.3	1.6	2	4.8	5	7

Оскільки заздалегідь невідомо, скільки разів треба повторювати проходження масивом, доцільно організувати ще один цикл. Зовнішній цикл (по i) дозволяє перебрати всі елементи масиву для порівняння з іншими елементами, доступ до яких організовано у вкладеному циклі (по j). Можна навести два різновиди організації порівнянь елементів у циклах:

1) Циклічно порівнюються два сусідні елементи – $a[j]$ та $a[j+1]$. За рахунок того, що після кожного проходження циклом один з елементів ставатиме на своє місце, можна скоротити кількість повторювань вкладеного циклу до $(n - i - 1)$ разів.

```
for (i=0; i<n-1; i++)
    for (j=0; j<n-i-1; j++)
        if (a[j] > a[j+1])
        {
```

```

temp = a[j];
a[j] = a[j+1];
a[j+1] = temp;
}

```

Перевіримо роботу цього циклу на масиві з 7-ми елементів:

9, 8, 7, 6, 5, 4, 3.

Для наочності взято початковий масив, елементи якого розташовано у порядку, протилежному до того, який має бути набуто. У таблиці подано вигляд масиву після кожної ітерації циклу по i . Сірим кольором і рамкою виокремлено елементи, які беруть участь у перестановках наступної ітерації.

	0	1	2	3	4	5	6
<i>початок</i>	9	8	7	6	5	4	3
$i=0$	8	7	6	5	4	3	9
$i=1$	7	6	5	4	3	8	9
$i=2$	6	5	4	3	7	8	9
$i=3$	5	4	3	6	7	8	9
$i=4$	4	3	5	6	7	8	9
$i=5$	3	4	5	6	7	8	9

2) Циклічно порівнюються два елементи – $a[i]$ та $a[j]$. За рахунок того, що після кожного проходження циклом один з елементів ставатиме на своє місце, можна скоротити кількість повторювань вкладеного циклу, розпочинаючи вкладений цикл не з 0-го елемента, а з $(i + 1)$ -го.

```

for (i=0; i<n-1; i++)
for (j=i+1; j<n; j++)
if (a[i] > a[j])
{
temp = a[i];
a[i] = a[j];
a[j] = temp;
}

```

Подібно до першого фрагмента, наведемо подібну таблицю для того самого початкового масиву.

	0	1	2	3	4	5	6
<i>початок</i>	9	8	7	6	5	4	3
$i=0$	3	9	8	7	6	5	4
$i=1$	3	4	9	8	7	6	5
$i=2$	3	4	5	9	8	7	6
$i=3$	3	4	5	6	9	8	7
$i=4$	3	4	5	6	7	9	8
$i=5$	3	4	5	6	7	8	9

Сортування елементів масиву організуємо в окремій функції. Слід звернути увагу на заголовок цієї функції:

```
void sort(float a[], int n)
```

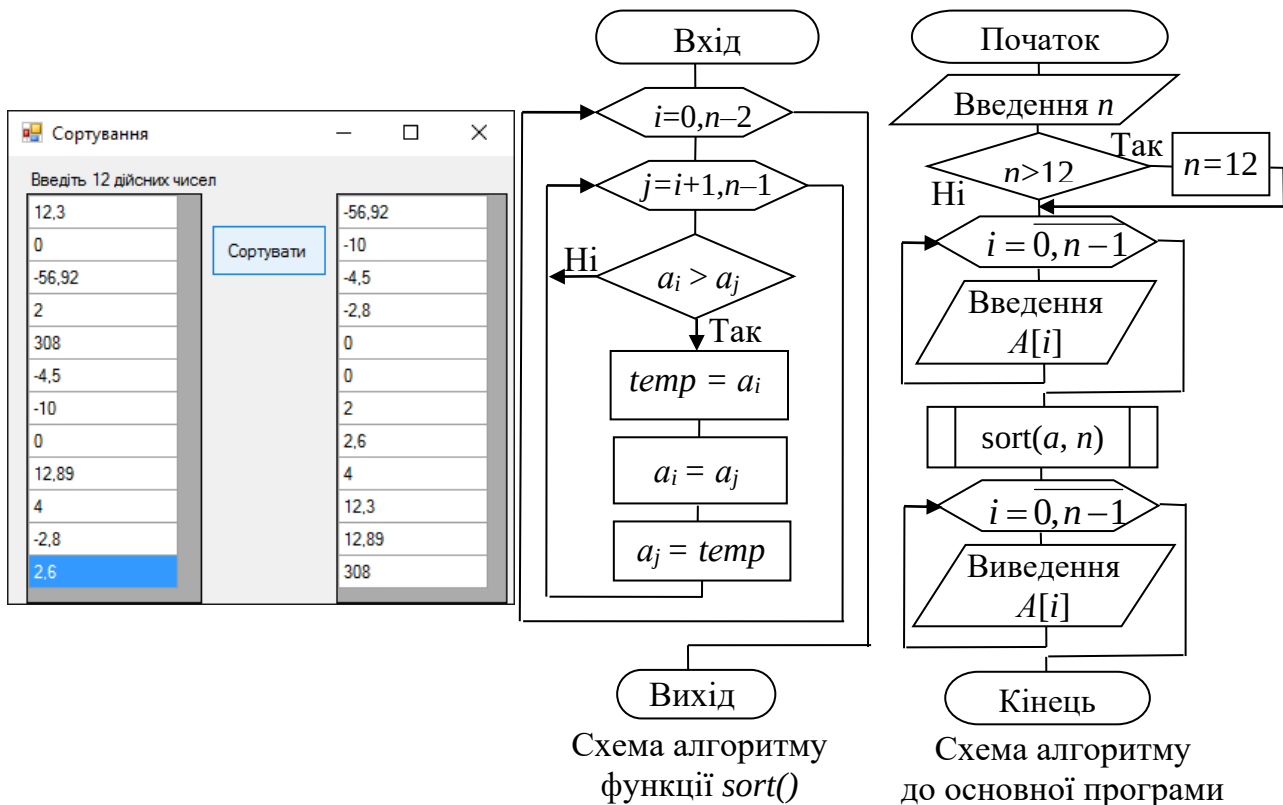
де `sort` – ім'я функції, за яким її можна буде викликати;

`float a[], int n` – параметри функції: масив дійсних чисел `a` (який функція сортуватиме) і кількість елементів масиву `n`;

`void` – тип результату, який вказує на те, що функція виконуватиме сортування і змінюватиме масив, але жодних інших обчислень вона не виконуватиме і нічого, окрім зміненого масиву, повертати немає потреби.

Викликати розглянуту функцію можна, якщо написати ім'я функції й у круглих дужках зазначити два фактичні параметри: ім'я масиву та кількість його елементів, наприклад:

```
sort(a, 12);           // Якщо елементів є 12
sort(a, n);            // Якщо елементів є n
```



Текст функції та основної програми:

```
void sort(float a[], int n)
{ int i, j; // Параметри наступних циклів
  float temp; // Тимчасова зміна для переставлянь елементів
  for (i=0; i<n-1; i++)
  for (j=i+1; j<n; j++)
    if (a[i] > a[j])
    { temp = a[i];
      a[i] = a[j];
      a[j] = temp;
    }
}
```

```

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{ double a[12];
  for (int i=0; i<12; i++)
    a[i]=Convert::ToDouble(dataGridView1->Rows[i]->Cells[0]->Value);
  sort(a, n); // Виклик функції
  for (int i=0; i<12; i++)
    dataGridView2->Rows[i]->Cells[0]->Value = a[i].ToString();
}
// При завантаженні форми створити по 12 рядків в обох dataGridView-ах
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{ dataGridView1->Rows->Add(12);
  dataGridView2->Rows->Add(12);
}

```

Другий спосіб

Аналізуючи попередній алгоритм сортування, можна скоротити кількість проходжень по масиву. В наведеному нижче прикладі функції sort2() запропоновано алгоритм сортування, де булева змінна ok відіграє роль своєрідного ліхтарика, який «вмикається» (ok=1) лише в разі переставлення сусідніх елементів, тобто є сигналом, що ще не всі елементи є упорядковано і треба повторювати перевірки.

Текст функції:

```

void sort2 (float a[], int n)
{ int i; float temp; bool ok=1;
  while (ok)
  { ok=0;
    for (j=0; j<n-1; j++)
      if (a[j] > a[j+1])
      { temp = a[j];
        a[j] = a[j+1];
        a[j+1] = temp;
        ok = 1;
      }
  }
}

```

Перевіримо роботу цього циклу на масиві з 7-ми елементів:

-0.9, -2.1, 6.3, 3, 8, 5.1, 30.

В таблиці показано вигляд масиву після кожного проходження масивом.

	0	1	2	3	4	5	6
початок	-0.9	-2.1	6.3	3	8	5.1	30
1-й прохід	-2.1	-0.9	3	6.3	5.1	8	30
2-й прохід	-2.1	-0.9	3	5.1	6.3	8	30
3-й прохід	-2.1	-0.9	3	5.1	6.3	8	30

Отже, на відміну від попереднього способу сортування, кількість проходжень циклом істотно скорочується, залежно від початкового розташування елементів. Останнє проходження циклом можна назвати холостим, оскільки перестановок у ньому немає, а отже, булева змінна `ok` не «вмикається» (`ok=1`), що є сигналом, що всі елементи упорядковано, – і відбувається вихід з циклу `while`.

Третій спосіб

Доволі поширеним є ще один спосіб сортування за допомогою мінімального елемента, алгоритм якого реалізовано у наведеній нижче функції `sort3()`. Його суть полягає в тому, що з послідовності вибирають мінімальний (чи максимальний) елемент і переставляють його до початку (кінця) масиву, змінюючи місцями з перевірюваним. Кількість перестановок – $(n+1)/2$.

Розглянемо цей алгоритм для сортування за зростанням масиву з 7-ми елементів:

5, -1.3, -7.1, 4.8, 2, 7, 1.6.

Щоб елементи було розташовано за зростанням, слід віднайти мінімальний елемент масиву і поставити його на перше місце. Після цього повторити аналогічні дії для решти елементів (окрім першого, який вже стоїть на своєму місці). У таблиці на кожній ітерації жирним позначено мінімальний елемент, рамкою виокремлено елементи, які беруть участь у подальшому пошуку мінімального елемента і перестановках.

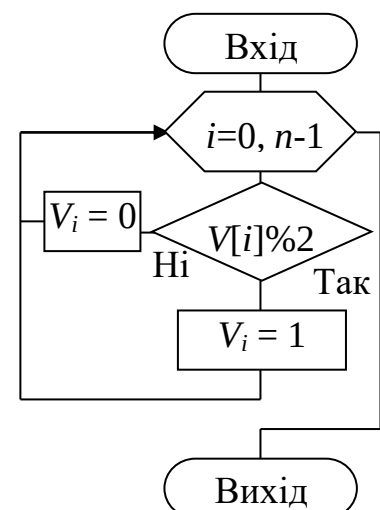
	0	1	2	3	4	5	6
<i>початок</i>	5	-1.3	-7.1	4.8	2	7	1.6
<i>i=0</i>	-7.1	-1.3	5	4.8	2	7	1.6
<i>i=1</i>	-7.1	-1.3	5	4.8	2	7	1.6
<i>i=2</i>	-7.1	-1.3	1.6	4.8	2	7	5
<i>i=3</i>	-7.1	-1.3	1.6	2	4.8	7	5
<i>i=4</i>	-7.1	-1.3	1.6	2	4.8	7	5
<i>i=5</i>	-7.1	-1.3	1.6	2	4.8	5	7

void sort3(float a[], int n) //n – кількість елементів масиву

```

{
    int i, j, imin; float temp;
    for (i=0; i<n-1; i++)
    {
        imin=i;
        for (j=i+1; j<n; j++)
            if (a[imin] > a[j]) imin=j;
        temp = a[i];
        a[i] = a[imin];
        a[imin] = temp;
    }
}

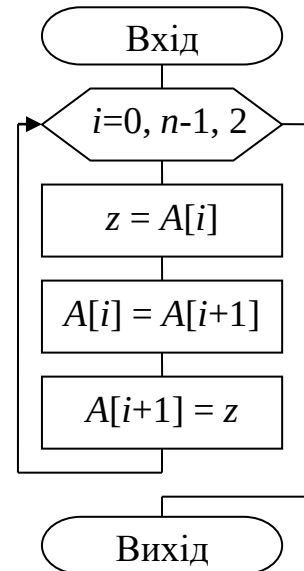
```



Приклад 5.15 В одновимірному масиві дійсних чисел, який складається з 14-ти (парної кількості) елементів, переставити місцями елементи, які стоять поряд (1 та 2, 3 та 4 тощо).

Текст функції та її виклику в основній програмі:

```
#include <iostream>
using namespace std;
void Perehod (double A[], int n)
{ int i; double z;
  for (i=0; i<n; i+=2)
    { z = A[i];
      A[i] = A[i+1];
      A[i+1] = z;
    }
}
int main()
{
  double a[14];
  cout<<"Введіть 14 чисел:\n";
  for (int i=0; i<14; i++) cin>>a[i];
  Perehod (a, 14);
  cout<<"Переставити місцями елементи, які стоять поряд (1 та 2, 3 та 4 тощо):\n";
  for (int i=0; i<14; i++)
    cout <<a[i] << " ";
  return 0;
}
```



Результати роботи:

Введіть 14 чисел:

-2.4 0 33 5 0 1.2 3 77 2 99 0 -4 4 14

Переставити місцями елементи, які стоять поряд (1 та 2, 3 та 4 тощо):

0 -2.4 5 33 1.2 0 77 3 99 2 -4 0 14 4

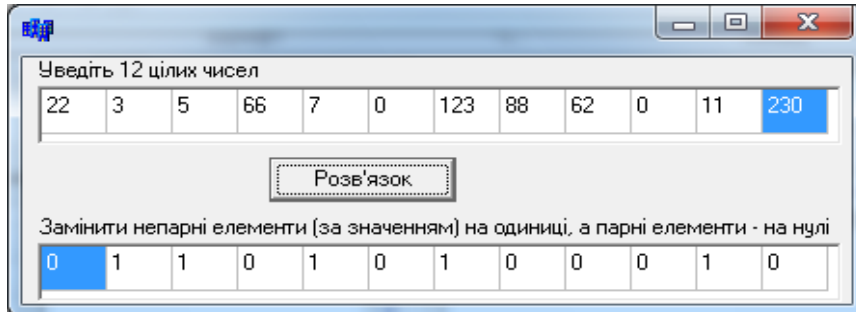
Приклад 5.16 У послідовності з 12-ти цілих чисел непарні елементи (за значенням, а не за індексом) замінити на одиниці, а парні елементи – на нулі.

Розв'язок. Введення та виведення послідовності чисел організуємо з елемента керування dataGridView, роботу якого докладно розглянуто у прикладі 5.7.

Текст функції та основної програми:

```
void Zamina(int V[], int n)
{ for (int i=0; i<n; i++)
  if (V[i] % 2) V[i]=1; else V[i]=0;
}
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
  dataGridView1->Rows->Add(1);
  dataGridView2->Rows->Add(1);
}
```

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    int a[12], i;
    for (i=0; i<12; i++) a[i] = Convert::ToInt32(dataGridView1[i,0]->Value);
    Zamina(a, 12);
    for (i=0; i<12; i++) dataGridView2[i,0]->Value = a[i].ToString();
}
```



5.3 Двовимірні масиви

5.3.1 Організація двовимірних масивів

Доволі часто буває недостатньо однієї вимірності масиву для зручного зберігання даних. Наприклад, якщо є потреба зберігати результати сесії цілої групи (30 студентів): оцінки з п'яти предметів. На папері ці результати зазвичай заносяться до таблиці, кожний рядок якої є оцінками одного студента по всіх предметах, а кожний стовпчик – оцінками групи з конкретного предмета:

Прізвище	Математика	Фізика	Інформатика	Історія	Фізкультура
Антонов	95	95	95	95	95
Бондарь	90	85	90	95	95
Василенко	60	75	65	80	80
...	...	...	...	...	...

Так само згадані дані можуть зберігатися і в масиві, де номери рядків означають номери студентів, а номери стовпчиків – номери предметів:

	0	1	2	3	4
0	95	95	95	95	95
1	90	85	90	95	95
2	60	75	65	80	80
...	...	...	...	...	...

Цей масив є двовимірний: номер рядка визначає оцінки кожного студента, а номер стовпчика – предмет. Якщо в одному масиві зберігати оцінки всіх груп разом, то це буде вже тривимірний масив, і елемент визначатиметься ще й номером групи.

Як зазначалось на початку розділу, вимірність масиву визначається кількістю індексів. Елементи одновимірного масиву (вектора) мають один індекс,

двовимірного масиву (матриці, таблиці) – два індекси: перший з них – номер рядка, другий – номер стовпчика. Кількість індексів у масивах є необмежена. При розміщуванні елементів масиву в пам'яті комп'ютера першою чергою змінюється крайній правий індекс, потім решта – справа наліво.

Багатовимірний масив оголошується у програмі в такий спосіб:

```
<тип> <ім'я> [ <розмір1> ] [ <розмір2> ] ... [ <розмірN> ];
```

Кількість елементів масиву дорівнює добуткові кількості елементів за кожним індексом. У прикладі

```
int a[3][4];
```

оголошено двовимірний масив з 3-х рядків та 4-х стовпчиків (12-ти елементів) цілого типу:

```
a[0][0], a[0][1], a[0][2], a[0][3],
a[1][0], a[1][1], a[1][2], a[1][3],
a[2][0], a[2][1], a[2][2], a[2][3];
```

Під масив надається пам'ять, потрібна для розташування усіх його елементів. Елементи масиву один за одним, з першого до останнього, запам'ятовуються у послідовно зростаючих адресах пам'яті так само, як і елементи одновимірного масиву. Наприклад, масив `int a[3][4]` зберігається у пам'яті у такий спосіб:

0	1	2	3	4	5	6	7	8	9	10	11
a ₀₀	a ₀₁	a ₀₂	a ₀₃	a ₁₀	a ₁₁	a ₁₂	a ₁₃	a ₂₀	a ₂₁	a ₂₂	a ₂₃
0-й рядок				1-й рядок				2-й рядок			

Масив оцінок з п'яти предметів для групи з 30-ти студентів можна оголосити, наприклад, у такий спосіб:

```
int ocinki[30][5];
```

Якщо цей масив зберігатиме оцінки з наведеної таблиці, то до оцінки з математики першого в групі студента з прізвищем Антонов можна звернутися у такий спосіб: `ocinki[0][0]`, до оцінки Бондаря з інформатики: `ocinki[1][2]`.

Наведемо ще кілька прикладів оголошення масивів:

```
short Mas1 [5][5]; // Матриця 5x5=25 елементів цілого типу
char Mas2 [10][3]; // Двовимірний масив з 10x3=30 елементів символьного типу
double Mas3 [4][5][4]; // Тривимірний масив з 4x5x4=80 елементів дійсного типу
```

При оголошенні масиву можна ініціалізувати початкові значення його елементів, причому необов'язково усіх, наприклад:

```
1) int w[3][3]={ { 2, 3, 4 }, { 3, 4, 8 }, { 1, 0, 9 } };
2) double C[4][3]={1.1, 2, 3, 3.4, 0.5, 6.8, 9.7, 0.9};
3) double C[4][3]={ {1.1, 2, 3}, {3.4, 0.5, 6.8}, { 9.7, 0.9} };
4) double C[4][3]={ {1.1, 2}, {3, 3.4, 0.5}, {6.8}, {9.7, 0.9} };
```

У першому прикладі оголошено й ініціалізовано масив цілих чисел `w[3][3]`. Елементам масиву присвоєно значення зі списку: `w[0][0]=2`, `w[0][1]=3`, `w[0][2]=4`, `w[1][0]=3` тощо. Списки, виокремлені у фігурні дужки, відповідають рядкам масиву.

Записи другого й третього прикладів є еквівалентні, і в них елементи останнього рядка не є ініціалізовані, тобто не є визначені. У таких випадках у числових масивах неініціалізовані елементи набувають значення 0.

Розташування елементів
у прикладах 2 та 3

1.1	2	3
3.4	0.5	6.8
9.7	0.9	0
0	0	0

Розташування елементів
у прикладі 4

1.1	2	0
3	3.4	0.5
6.8	0	0
9.7	0.9	0

Багатовимірні масиви можна оголошувати й зі створюванням типу користувача (докладніше див. підрозд. 10.1):

```
typedef <тип_даних> <і'мя_типу>[<розмір1>][<розмір2>]...[<розмір n>];
<і'мя_типу> <і'мя_масиву>;
```

Наприклад, створимо тип з ім'ям `matr` як масив додатних цілих чисел з 10-ти рядків і 7-ми стовпчиків та оголосимо два масиви – `M1` і `M2` – створеного типу:

```
typedef unsigned short matr[10][7];
matr M1, M2;
```

Для оголошення масивів можна також використовувати типізовані констант-масиви, які дозволяють водночас оголосити масив і задати його значення в розділі оголошень констант, наприклад:

```
const int arr[2][5] = {{9, 3, 0, 12, -5},{ -7, 23, 2, 4, 0}};
```

але змінювати значення елементів констант-масивів у програмі є неприпустимо.

В C++ можна використовувати *перетин масиву* (section). Це поняття подібно до поняття мінору матриці в математиці. Перетин формується внаслідок вилучення однієї чи кількох пар квадратних дужок. Пари квадратних дужок можна відкидати лише справа наліво й лише послідовно. Перетини масивів використовуються при організації обчислювального процесу в функціях мовою C++, розроблюваних користувачем.

Приклади:

1) `int x[5][3];`

Якщо при звертанні до певної функції написати `x[0]`, то передаватиметься нульовий рядок масиву `x`.

2) `int y[2][3][4];`

При звертанні до масиву `y` можна записати, наприклад, `y[1][2]` – і буде передаватися вектор з чотирьох елементів, а звертання `y[1]` надасть двовимірний масив розміром 3×4 . Не можна писати `y[2][4]`, вважаючи на увазі, що передаватиметься вектор, тому що це не відповідає обмеженню, накладеному на використання перетинів масиву.

5.3.2 Введення-виведення двовимірних масивів

Здійснювати введення-виведення значень елементів масиву можна лише поелементно, для чого слід організовувати цикли, в яких послідовно змінюватимуться значення індексів елементів.

Введення двовимірних масивів у консолі

Введення у консолі матриці а розміром 3×4:

1 *спосіб* (див. приклад 5.17):

```
for(i=0; i<3; i++)
for(j=0; j<4; j++)
{ cout<<"Введіть елемент "<<i+1<<"-го рядка "<<j+1<<"-го стовпця: ";
  cin>>a[i][j];
}
```

2 *спосіб* (див. приклад 5.18):

```
for(i=0; i<3; i++)
{ cout<<"Введіть 4 елементи "<<i+1<<"-го рядка "<<endl;
  for(j=0; j<4; j++) cin>>a[i][j];
}
```

3 *спосіб* (див. приклад 5.19):

```
cout<<"Введіть матрицю з 3-х рядків і 4-х стовпців:"<<endl;
for(i=0; i<3; i++)
for(j=0; j<4; j++) cin>>a[i][j];
```

Виведення двовимірних масивів у консолі

Для виведення у консолі двовимірного масиву а розміром 3×4 у вигляді матриці елементи слід розмістити у рівні стовпці. Це можна зробити за допомогою символу табуляції `"\t"`:

```
for (i=0; i<3; i++)
{ for (j=0; j<4; j++) cout << a[i][j] << "\t";
  cout << endl;
}
```

При виведенні елементів матриць дійсних чисел командою `cout<<` доречним є використання маніпуляторів форматування:

- **setprecision**, який обмежує кількість знаків після десяткової крапки;
- **fixed**, який задає формат з фіксованою крапкою;
- **setw**, який дозволяє задавати ширину (мінімальну кількість символівних позицій) кожного виведеного числа. Якщо виведене число має ширину меншу, аніж зазначену у модифікаторі `setw`, перед ним будуть виведені пробіли:

```
for (i=0; i<3; i++)
{ for (j=0; j<4; j++)
  cout<<fixed<<setprecision(2)<<setw(5)<<a[i][j]/3.<<"\t";
  cout<<endl;
}
```

Для можливості використання цих маніпуляторів слід долучити заголовний файл `iomanip` командою `#include <iomanip>`.

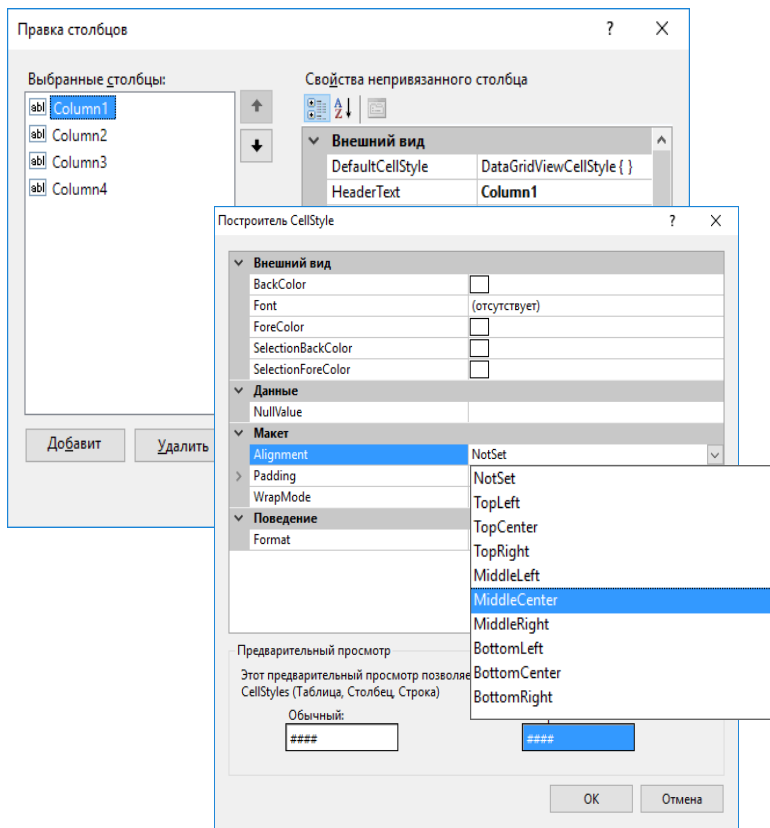
Введення елементів двовимірних масивів з dataGridView на формі

Елемент DataGridView має вигляд таблиці з клітинками, в кожній з яких можна розмістити величину рядкового типу (String^). Щоб звернутись до значення Value конкретної клітинки слід зазначити номер рядка Rows[i] і номер стовпця Cells[j]: dataGridView1->Rows[i]->Cells[j]->Value.

Після розміщення на формі елемента DataGridView через діалогове вікно *DataGridView Задачі*, яке автоматично з'явиться, слід командою *Додати стовбець* утворити чотири стовпці, натиснувши чотири рази кнопку *Додати* (див. с. 153).

Програмно створити стовпці можна такою командою:

```
dataGridView1->ColumnCount = 4;
```



За допомогою команди *Правка стовбцов* діалогового вікна *DataGridView Задачі* можна налаштувати параметри кожного зі стовпців. Наприклад, можна змінювати ширину (властивість Width), задавати вирівнювання вмісту клітинок стовпця (властивість DefaultCellStyle->Alignment), наприклад, по центру (значення MiddleCenter), вписати надпис стовпця (властивість HeaderText), приміром, значення "1-й стовпець", тощо.

Також для DataGridView у властивості AllowUserToAddRows (дозвіл на автоматичне долучення рядків) доцільно задати значення False.

Створювати (додавати) нові рядки будемо у програмному коді, створивши функцію Form1_Load і вписавши в неї команду створення трьох рядків:

```
dataGridView1->Rows->Add(3);
```

або

```
dataGridView1->RowCount = 3;
```

Також під час створення форми створимо заголовки рядків та стовпців:

```
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
    dataGridView1->Rows->Add(3);
    dataGridView1->RowHeadersWidth = 90;
    for(int i=0; i<3; i++)
        dataGridView1->Rows[i]->HeaderCell->Value = Convert::ToString(i+1)+"-й рядок";
    for(int j=0; j<4; j++)
        dataGridView1->Columns[j]->HeaderText = Convert::ToString(j+1)+"-й стовпець";
}
```

	1-й стовпець	2-й стовпець	3-й стовпець	4-й стовпець
► 1-й рядок	11	10	13	1
2-й рядок	38	30	30	29
3-й рядок	21	22	48	47

Для того щоб прибрати заголовки стовпців і рядків, слід задати значення False для властивостей ColumnHeadersVisible (заголовок стовпця) та RowHeadersVisible (стовпець із заголовками рядків). Тоді на формі такий матиме вигляд:

2	17	23	37
26	33	21	46
21	33	34	26

Введення матриці дійсних чисел а розміром 3×4 з dataGridView:

```
double a[3][4];
for (int i=0; i<3; i++)
for (int j=0; j<4; j++)
    a[i][j]=Convert::ToDouble(dataGridView1->Rows[i]->Cells[j]->Value);
    або
for (int i=0; i<3; i++)
for (int j=0; j<4; j++)
    a[i][j]=Convert::ToDouble(dataGridView1[j,i]->Value);
```

Можна додати перевірку на незаповнені комірки:

```
for (int i=0; i<3; i++)
for (int j=0; j<4; j++)
    if (dataGridView1[j,i]->Value==nullptr)
    { MessageBox::Show("Заповніть усі комірки!");
      return;
    }
else
    a[i][j]=Convert::ToDouble(dataGridView1[j,i]->Value);
```

Виведення елементів одновимірних масивів до різних елементів керування

Виведення матриці а розміром 3×4 на форму до dataGridView:

```
for (int i=0; i<3; i++)
for (int j=0; j<4; j++)
    dataGridView1[j,i]->Value = Convert::ToString(a[i][j]);
```

Очищення dataGridView:

```
for (int i=0; i<3; i++)
for (int j=0; j<4; j++) dataGridView1[j,i]->Value = "";
```

Заповнення dataGridView випадковими числами:

```
Random^ rnd = gcnew Random();
for (int i=0; i<3; i++)
for (int j=0; j<4; j++)
    dataGridView1->Rows[i]->Cells[j]->Value = rnd->Next(50);
```

У цьому прикладі засобами методу Next(50) усі комірки dataGridView

буде заповнено випадковими цілими числами від 0 до 49. Для методу `Next` можна задавати межі діапазону. Так, метод `Next(-100,101)` генеруватиме ціле число у діапазоні від -100 до 100. Якщо ж використати ділення цього числа на дійсне значення 10.0, то можна буде заповнити комірки випадковими дійсними числами від -10 до 10:

```
Random^ rnd = gcnew Random();
for (int i=0; i<3; i++)
for (int j=0; j<4; j++)
    dataGridView1[j,i]->Value = Convert::ToString(rnd->Next(-100,101)/10.);
```

Крім методу `Next()`, для типу `Random` існує метод `NextDouble()`, який генерує випадкове число в діапазоні від 0 до 1.

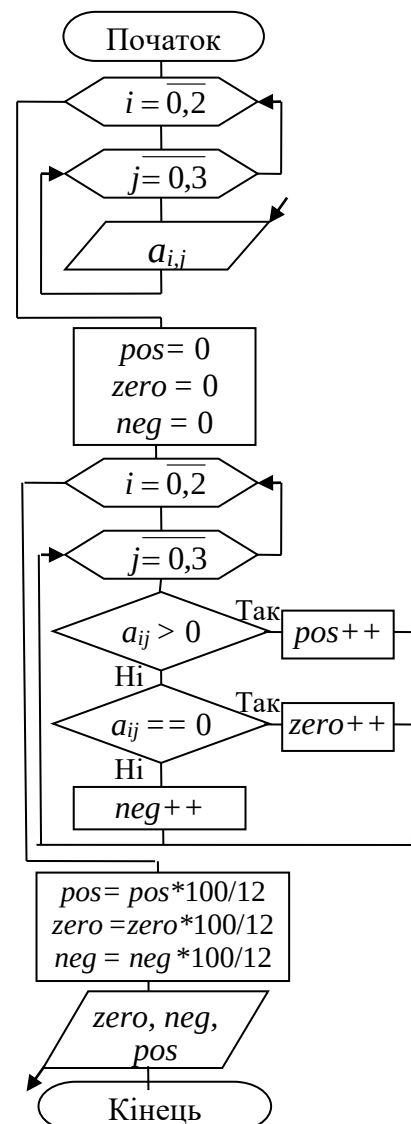
5.3.3 Програмування базових алгоритмів опрацювання двовимірних масивів

Приклад 5.17. Ввести матрицю 3×4 цілих чисел та обчислити відсотковий вміст від'ємних, нульових і додатних елементів.

Текст консольної програми та схема алгоритму:

```
#include <iostream>
// бібліотека для використання маніпуляторів форматування
#include <iomanip>
using namespace std;

int main()
{ setlocale(0, ".1251");
  int a[3][4], i, j;
  for (i=0; i<3; i++)
  for (j=0; j<4; j++)
  { cout<<"Введіть елемент "<<i+1<<"-го рядка "
    <<j+1<<"-го стовпця: ";
    cin>>a[i][j];
  }
  double pos, zero, neg;
  pos = zero = neg = 0;
  for (i=0; i<3; i++)
  for (j=0; j<4; j++)
  { if (a[i][j]>0) pos++;
    else
      if (a[i][j] == 0) zero++;
      else neg++;
  }
  pos *= 100.0/12;
  neg *= 100.0/12;
  zero *= 100.0/12;
  cout<<"\nВідсотковий вміст від'ємних елементів - "
    <<setprecision(3) <<neg<<" %"<<endl;
```



```

cout<<"Відсотковий вміст нульових елементів - "<< zero<<" %"<<endl;
cout<<"Відсотковий вміст додатних елементів - "<< pos<<" %"<<endl;
system ("pause>>void");
return 0;
}

```

Результати роботи консольної програми:

```

Введіть елемент 1-го рядка 1-го стовпця: 23
Введіть елемент 1-го рядка 2-го стовпця: 4
Введіть елемент 1-го рядка 3-го стовпця: 6
Введіть елемент 1-го рядка 4-го стовпця: 0
Введіть елемент 2-го рядка 1-го стовпця: -4
Введіть елемент 2-го рядка 2-го стовпця: -8
Введіть елемент 2-го рядка 3-го стовпця: 23
Введіть елемент 2-го рядка 4-го стовпця: 2
Введіть елемент 3-го рядка 1-го стовпця: 0
Введіть елемент 3-го рядка 2-го стовпця: -12
Введіть елемент 3-го рядка 3-го стовпця: 1
Введіть елемент 3-го рядка 4-го стовпця: 4

Відсотковий вміст від'ємних елементів - 25 %
Відсотковий вміст нульових елементів - 16.7 %
Відсотковий вміст додатних елементів - 58.3

```

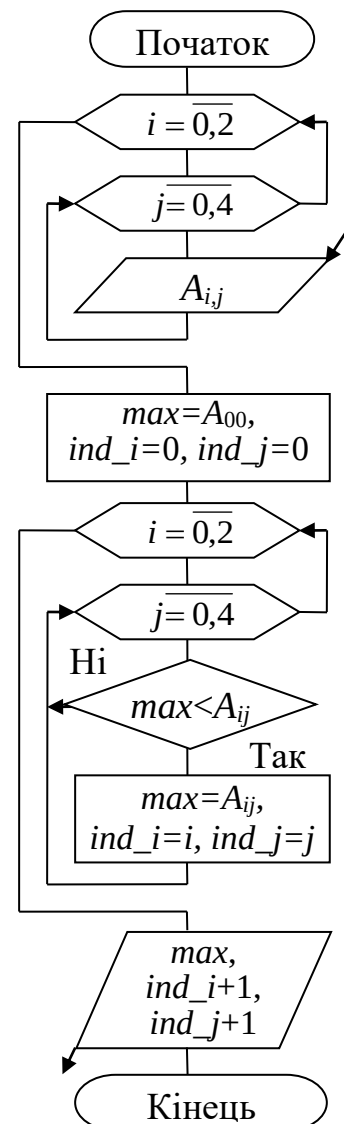
Приклад 5.18. Скласти схему алгоритму і розробити проект для введення матриці дійсних чисел розмірністю 3×5 і віднайти максимальний елемент матриці та його індекси.

Текст консольної програми та схема алгоритму:

```

#include <iostream>
using namespace std;
int main()
{ setlocale(0, ".1251");
  double A[3][5], max;  int i, j, ind_i, ind_j;
  for(i=0; i<3; i++)
  { cout << "Введіть 5 елементів " << i+1
    << "-го рядка "<<endl;
    for (j=0; j<5; j++) cin >> A[i][j];
  }
  max = A[0][0];  ind_i = 0; ind_j = 0;
  for (i=0; i<3; i++)
  for (j=0; j<5; j++)
    if (max < A[i][j])
    { max = A[i][j];
      ind_i = i; ind_j = j;
    }
  cout << "\nmax = " << max << "      " << ind_i+1
    << "-й рядок i "<< ind_j + 1 << "-й стовпчик";
  system ("pause>>void");
  return 0;
}

```



Результати роботи консольної програми:

Введіть 5 елементів 1-го рядка

-2.4 0 33 5 0

Введіть 5 елементів 2-го рядка

1.2 3 77 2 99.6

Введіть 5 елементів 3-го рядка

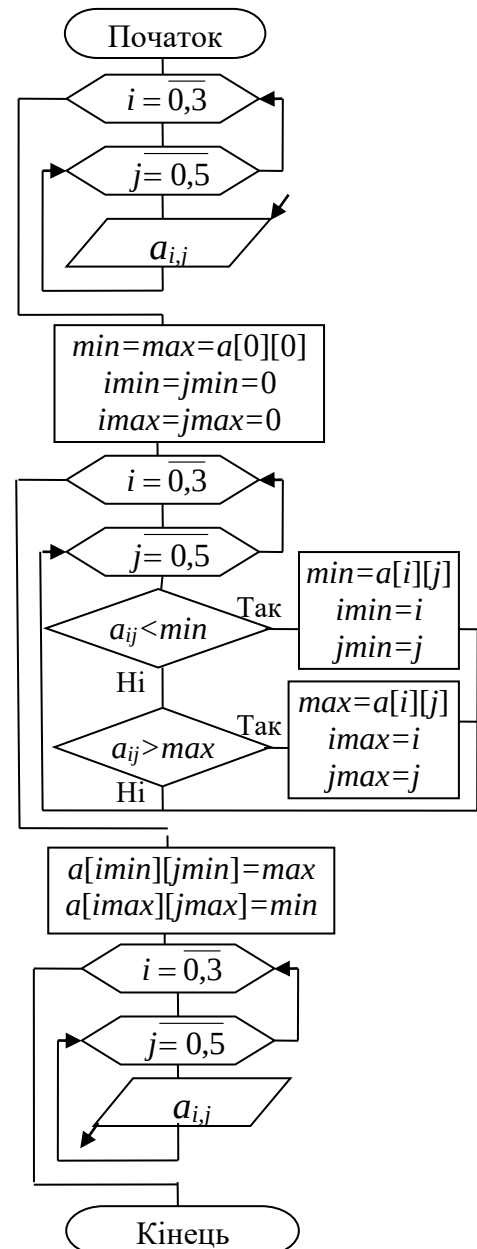
0 -4 4 14 -40

max = 99.6 2-й рядок і 5-й стовпчик

Приклад 5.19. Ввести матрицю 4×6 дійсних чисел і поміняти місцями максимальний і мінімальний елементи.

Програмний код та схема алгоритму:

```
#include <iostream>
using namespace std;
int main()
{
    setlocale(0, ".1251");
    double a[4][6], min, max;
    int i, j, imin, jmin, imax, jmax;
    cout << "Введіть матрицю з 4-х рядків і 6-ти стовпців:" << endl;
    for (i=0; i<4; i++)
        for (j=0; j<6; j++) cin >> a[i][j];
    min=max=a[0][0];
    imin=jmin=imax=jmax=0;
    for (i=0; i<4; i++)
        for (j=0; j<6; j++)
        {
            if (a[i][j]<min)
            {
                min=a[i][j];
                imin=i; jmin=j;
            }
            if (a[i][j]>max)
            {
                max=a[i][j];
                imax=i; jmax=j;
            }
        }
    a[imin][jmin]=max;
    a[imax][jmax]=min;
    cout << "\nМатриця, в якій поміняні місцями максимальний і мінімальний елементи:" << endl;
    for (i=0; i<4; i++)
    {
        for (j=0; j<6; j++) cout << a[i][j] << "\t";
        cout << endl;
    }
    system("pause");
    return 0;
}
```



Результати роботи консольної програми:

Введіть матрицю з 4-х рядків і 6-ти стовпців:

23	0	-4	9	2.6	8.1
1	-100	4.6	7	0	9
-3	0	5.7	-0.6	34	10
1	2	3	-4	0	59

Матриця, в якій поміняні місцями максимальний і мінімальний елементи:

23	0	-4	9	2.6	8.1
1	59	4.6	7	0	9
-3	0	5.7	-0.6	34	10
1	2	3	-4	0	-100

Приклад 5.20. За елементами матриці розміром 5×10 цілих чисел сформувати вектор максимальних елементів стовпців матриці.

Розв'язок. Налаштувати форму рекомендуємо у такій послідовності.

1) Розмістити на формі перший елемент `dataGridView` і сформувати (команда *Додати стовбець* в контекстному меню) на ньому 10 стовпців, для кожного з яких командою контекстного меню *Правка стовбців* задати для властивості `Width` (ширина стовпця) значення 40 пікселів (замість 100).

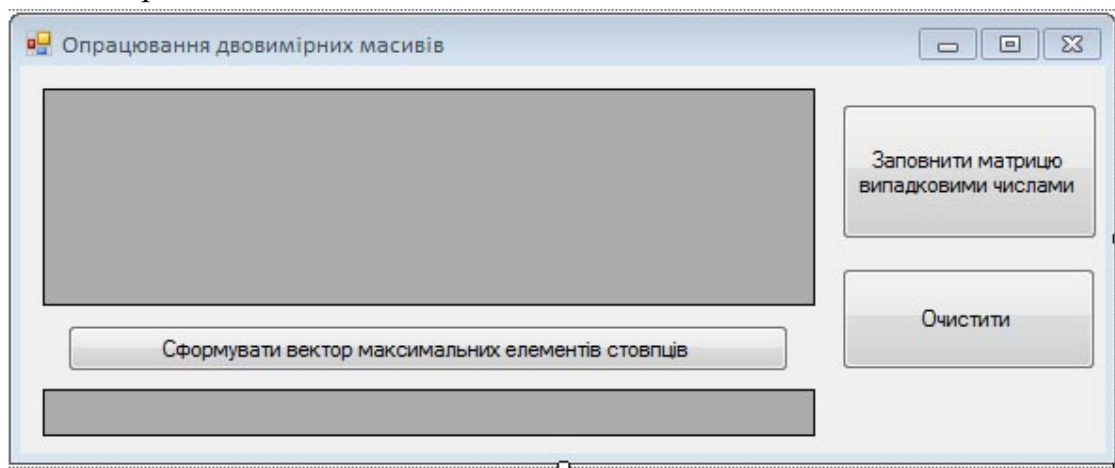
2) Задати для властивості `AllowUserToAddRows` (дозвіл автоматичного додавання рядків) значення `False`.

3) Задати для властивостей `RowHeadersVisible` і `ColumnHeadersVisible` (відображення заголовка рядків і стовпців) значення `False`.

4) Відредагувати розміри (властивість `Size`) на значення 410; 115.

5) Скопіювати (`[Ctrl]+[C]`) налаштований `dataGridView1`, виконати команду вставлення – `[Ctrl]+[V]` (при цьому автоматично створиться `dataGridView2` з уже встановленими властивостями). Можна перемістити `dataGridView2` під `dataGridView1` і зменшити розмір `dataGridView2` (властивість `Size`) до 410; 25.

6) Розмістити на формі три кнопки `button` та задати надписи на них (властивість `Text`): для `button1` – *Сформувати вектор максимальних елементів стовпців*, для `button2` – *Заповнити матрицю випадковими числами*; для `button3` – *Очистити*. Задати надпис на формі (властивість `Text`) – *Опрацювання двовимірних масивів*.



Подвійним клацанням по відповідних елементах на формі створити функції Form1_Load, button1_Click, button2_Click і button3_Click та вписати в них програмний код.

Текст програмного коду:

```
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{ dataGridView1->Rows->Add(5); dataGridView2->Rows->Add();
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    int a[5][10], x[10];
    for (int i=0; i<5; i++)
    for (int j=0; j<10; j++)
        a[i][j]=Convert::ToInt32(dataGridView1->Rows[i]->Cells[j]->Value);
    for (int j=0; j<10; j++)
    { x[j]=a[0][j];
      for (int i=0; i<5; i++)
          if(x[j]<a[i][j]) x[j]=a[i][j];
      dataGridView2->Rows[0]->Cells[j]->Value = Convert::ToString(x[j]);
    }
}
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e)
{ Random^ rnd = gcnew Random();
  for (int i=0; i<5; i++)
  for (int j=0; j<10; j++) dataGridView1[j,i]->Value = rnd->Next(50);
}
private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e)
{ for (int i=0; i<5; i++)
  for (int j=0; j<10; j++) dataGridView1[j,i]->Value = "";
  for (int j=0; j<10; j++) dataGridView2[j,0]->Value = "";
}
```

Результати роботи:

Опрацювання двовимірних масивів

1	14	50	19	21	34	2	6	44	44
22	33	18	9	22	2	12	11	12	33
7	3	27	19	6	49	10	24	33	0
15	42	16	45	30	41	34	0	23	28
0	23	0	40	25	40	37	15	49	11

Заповнити матрицю випадковими числами

Очистити

Сформувати вектор максимальних елементів стовпців

22	42	50	45	30	49	37	24	49	44

Приклад 5.21. Ввести матрицю цілих чисел розмірністю 5×7 та обчислити елементи вектора сум від'ємних елементів стовпців матриці.

Програмний код в онлайн-компіляторі та схема алгоритму:

```
#include <iostream>
using namespace std;
int main()
{ int a[5][7], x[7], i, j;
  cout << "Введіть матрицю з 5-ти рядків
           і 7-ми стовпців:"<<endl;
  for (i=0; i<5; i++)
  for (j=0; j<7; j++) cin>>a[i][j];
  cout << "Вектор сум від'ємних елементів
           стовпців матриці:\n";
  for (j=0; j<7; j++)
  { x[j]=0;
    for (i=0; i<5; i++)
      if (a[i][j]<0) x[j] += a[i][j];
    cout << x[j] << " ";
  }
  return 0;
}
```

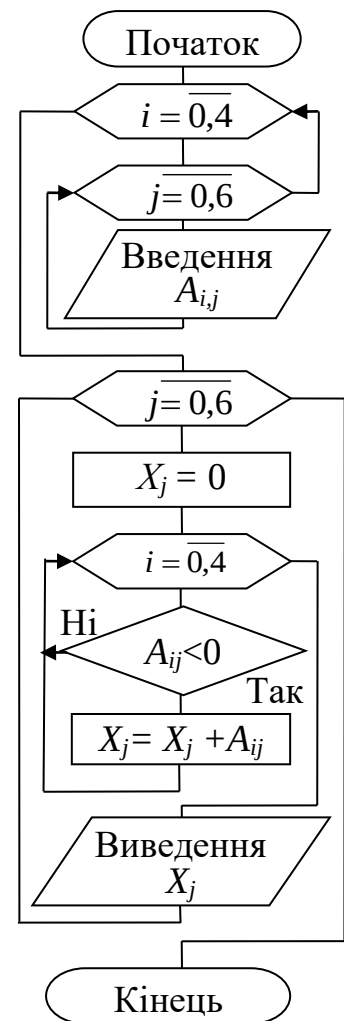
Результати роботи консольної програми:

Введіть матрицю з 5-ти рядків і 7-ми стовпців:

-24	0	33	5	0	-6	-7
12	-3	77	-2	9	19	6
0	-4	4	-14	-40	-2	2
1	2	3	4	5	6	-7
2	25	2	3	-3	-1	-1

Вектор сум від'ємних елементів стовпців матриці:

-24	-7	0	-16	-43	-9	-15



Приклад 5.22. З елементів матриці розміром 4×7 цілих чисел сформувати вектор середніх арифметичних парних елементів непарних (1, 3, 5 і 7) стовпців матриці.

Розв'язок. Оскільки при формуванні вектора його індекси не збігаються з індексами непарних стовпців матриці, за якими слід формувати вектор, виникає потреба у використанні додаткової змінної, наприклад, k для індексів вектора.

Текст консольної програми та схема алгоритму:

```
#include <iostream>
using namespace std;
int main()
{
  setlocale(0, ".1251");
  int i, j, a[4][7]; double v[4];
  cout << "Введіть матрицю з 4-х рядків і 7-ми стовпців:"<<endl;
  for (i=0; i<4; i++)
  for (j=0; j<7; j++) cin>>a[i][j];
  for (j=0; j<7; j++)
  { double sum = 0;
    for (i=0; i<4; i++)
      if (j % 2 == 1) sum += a[i][j];
    v[j/2] = sum / 2;
  }
  for (k=0; k<4; k++)
    cout << v[k] << " ";
  return 0;
}
```

```

for (i=0; i<4; i++)
for (j=0; j<7; j++) cin>>a[i][j];
cout<<"\nВектор середніх арифметичних парних
елементів непарних (1, 3, 5 та 7) стовпців:"<<endl;
int kol, k=0;
for (j=0; j<7; j+=2)
{ kol = 0;
  v[k] = 0;
  for (i=0; i<4; i++)
    if (a[i][j]%2 == 0 && a[i][j])
      { v[k] += a[i][j];
        kol++;
      }
  if (kol) v[k] /= kol;
  cout<< v[k]<<"\t\t";
  k++;
}
cout<<endl;
system ("pause>>void");
return 0;
}

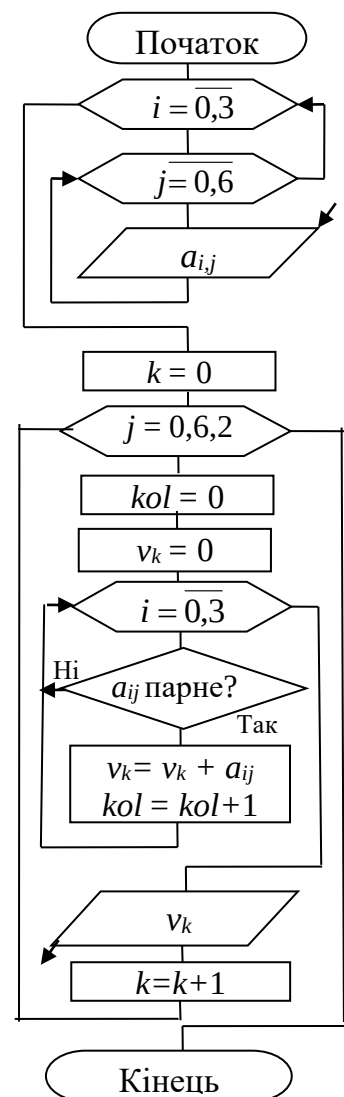
```

Результати роботи програми:

Введіть матрицю з 4-х рядків і 7-ми стовпців:

1	3	-6	0	9	61	8
2	8	-4	45	3	-1	0
4	-45	13	4	13	8	4
20	6	1	2	7	5	1

Вектор середніх арифметичних парних елементів непарних (1, 3, 5 та 7) стовпців:
 8.66667 -5 0 6



Приклад 5.23. З елементів матриці розмірністю 4×7 цілих чисел обчислити вектор добутків ненульових елементів парних (2, 4 та 6) стовпців матриці.

Текст програми з формою та блок-схема:

```

// Завантаження форми
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
  // Заповнення DataGridView стовпцями
  dataGridView1->ColumnCount = 7;
  dataGridView2->ColumnCount = 3;
  // Заповнення DataGridView рядками
  dataGridView1->RowCount = 4;
  dataGridView2->RowCount = 1;
  // Ширина заголовка рядків
  dataGridView1->RowHeadersWidth = 90;
  dataGridView2->ColumnHeadersVisible = false;
  dataGridView2->RowHeadersVisible = false;
}

```

```

dataGridView1->AllowUserToAddRows = false;
// Підписи у заголовках рядків і стовпців
for(int i=0; i<4; i++)
    dataGridView1->Rows[i]->HeaderCell->Value =
        Convert::ToString(i+1)+"-й рядок";
for(int j=0; j<7; j++)
{ dataGridView1->Columns[j]->HeaderText =
    Convert::ToString(j+1)+"-й стовпець";
    // Ширина стовпців
    dataGridView1->Columns[j]->Width = 60;
} }

```

```

// Вектор добутків ненульових елементів парних стовпців матриці
private: System::Void button1_Click(System::Object^ sender,
System::EventArgs^ e)
{

```

```

    int C[4][7], p[3], i, j, k=0;
    for (i=0; i<4; i++)
    for (j=0; j<7; j++)
        if (dataGridView1[j,i]->Value==nullptr)
        { MessageBox::Show("Заповніть ["+(i+1) + ", "
            + (j+1) + "] елемент");
            return;
        }
    else
        C[i][j]=Convert::ToInt32(dataGridView1[j,i]->Value);
    for (j=0; j<7; j++)
        if (j%2==1)
        { p[k]=1;
            for (i=0; i<4; i++)
                if (C[i][j]!=0) p[k] *= a[i][j];
            dataGridView2[k,0]->Value = p[k].ToString();
            k++;
        }
} }

```

```

// Заповнення DataGridView випадковими числами

```

```

private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e)
{

```

```

    Random^ rnd = gcnew Random();
    for (int i=0; i<4; i++)
    for (int j=0; j<7; j++)
        dataGridView1->Rows[i]->Cells[j]->Value = rnd->Next(-10,10);
}

```

```

// Очищення DataGridView

```

```

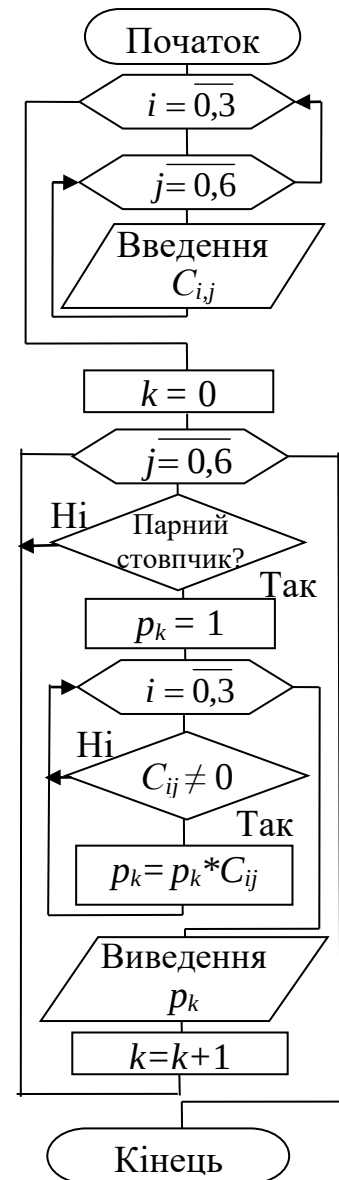
private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e)
{

```

```

    for (int i=0; i<4; i++)
    for (int j=0; j<7; j++) dataGridView1[j,i]->Value = "";
    for (int i=0; i<3; i++) dataGridView2[j,i]->Value = "";
}

```



Форма з результатами роботи:

The application window displays a 7x7 matrix with the following values:

	1-й стовпець	2-й стовпець	3-й стовпець	4-й стовпець	5-й стовпець	6-й стовпець	7-й стовпець
1-й рядок	-7	1	8	0	2	8	7
2-й рядок	4	-4	-8	5	-3	1	3
3-й рядок	3	5	-10	2	-3	1	-3
4-й рядок	0	0	-2	3	1	2	7

Buttons on the left: "Заповнити випадковими числами" (Fill with random numbers) and "Очистити" (Clear).

Below the matrix, a label reads: "Вектор добутків ненульових елементів парних (2, 4 та 6) стовпців матриці" (Vector of products of non-zero elements of even columns (2, 4 and 6) of the matrix).

The vector values are displayed in a table:

-20	30	16	

Приклад 5.24. З елементів матриці розмірністю 7×7 цілих чисел обчислити вектор скалярних добутків елементів першого рядка на стовпці матриці.

Розв'язок. Нагадаємо, що скалярний добуток – це сума добутків відповідних елементів. У даному разі:

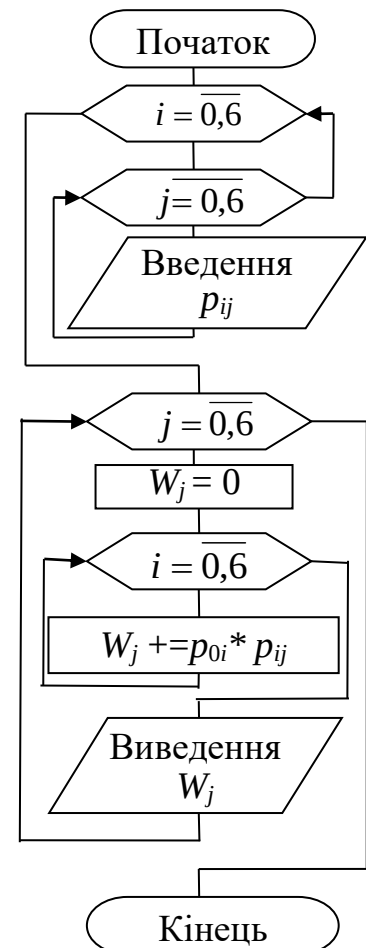
$$W[0] = p[0][0] \cdot p[0][0] + p[0][1] \cdot p[1][0] + p[0][2] \cdot p[2][0] + p[0][3] \cdot p[3][0] + \dots$$

$$W[1] = p[0][0] \cdot p[0][1] + p[0][1] \cdot p[1][1] + p[0][2] \cdot p[2][1] + p[0][3] \cdot p[3][1] + \dots$$

$$W[2] = p[0][0] \cdot p[0][2] + p[0][1] \cdot p[1][2] + p[0][2] \cdot p[2][2] + p[0][3] \cdot p[3][2] + \dots$$

Текст консольної програми та блок-схема:

```
#include <iostream>
using namespace std;
int main()
{ system("color F0");
  setlocale(0, ".1251");
  const int n=5;
  int i, j, p[n][n], W[n];
  cout<<" Введіть матрицю 5x5:"<<endl;
  for (i=0; i<n; i++)
  for (j=0; j<n; j++) cin>>p[i][j];
  cout<<" Вектор скалярних добутків елементів
    \n 1-го рядка на стовпці матриці:"<<endl;
  for (j=0; j<n; j++)
  { W[j]=0;
    for (i=0; i<n; i++)
      W[j] += p[0][i] * p[i][j];
    cout<< W[j]<<"\t";
  }
  cout<<endl;
  system ("pause>>void");
  return 0;
}
```



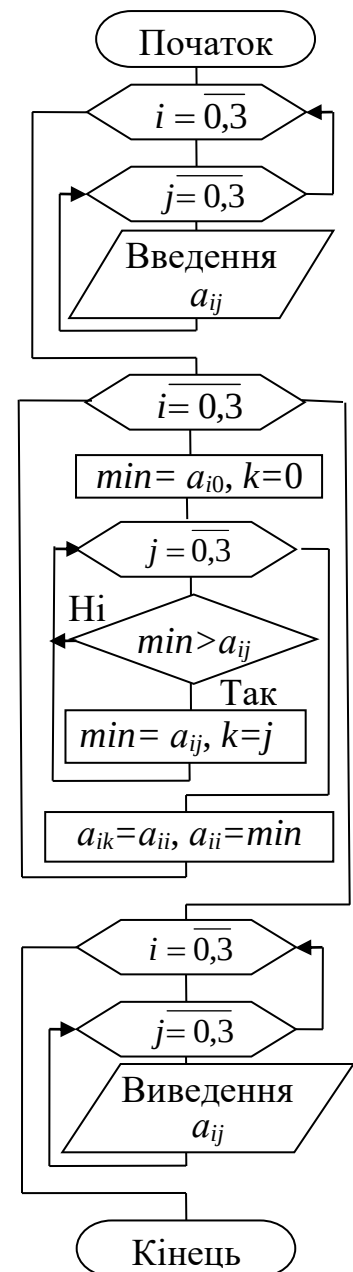
Результати роботи:

Введіть матрицю 5x5:				
2	3	1	0	-2
1	-2	0	5	12
0	1	5	-2	0
6	-1	7	3	1
3	0	2	0	-1
Вектор скалярних добутків елементів 1-го рядка на стовпці матриці:				
1	1	3	13	34

Приклад 5.25. Ввести матрицю розмірністю 4×4 дійсних чисел і замінити елементи головної діагоналі на мінімальні елементи відповідних рядків.

Текст консольної програми:

```
#include <iostream>
using namespace std;
int main()
{
    const int n=4;
    system("color F0");
    setlocale(0, ".1251");
    int i, j, a[n][n], min, k;
    cout<<"Введіть матрицю 4x4:"<<endl;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++) cin>>a[i][j];
    for (i=0; i<n; i++)
    { min=a[i][0]; // Запам'ятати перший елемент рядка
      k=0;        // та його індекс як мінімальний.
      for (j=0; j<n; j++) // Рухаючись по стовпчиках
          if (a[i][j]<min) // перевірити: якщо елемент
              { min=a[i][j]; // є менший за min,
                k=j; // запам'ятати його значення та індекс.
              }
      // Замінити мінімальний елемент рядка на елемент
      a[i][k]=a[i][i]; // діагоналі, а елемент діагоналі -
      a[i][i]=min;     // на мінімальний елемент.
    }
    cout<<"Матриця, в якій елементи головної діагоналі \n
           поміняли з мінімальним елементом \n відповідного
           рядка:" <<endl;
    for (i=0; i<n; i++)
    { for (j=0; j<n; j++)
        cout<< a[i][j]<<"\t";
      cout<<endl;
    }
    system ("pause>>void");
    return 0;
}
```



Результати роботи:

```

Введіть матрицю 4x4:
22      3      0      -5
-4      0      37     0
5       6      0      1
-5      4     -6      2
Матриця, в якій елементи головної діагоналі
поміняли з мінімальним елементом
відповідного рядка:
-5      3      0      22
0      -4     37     0
5       6      0      1
-5      4      2     -6
  
```

Приклад 5.26. Ввести дві цілочисельні матриці 3×3 й обчислити добуток цих матриць.

Розв'язок. Кожний елемент матриці-результату обчислюється за таким алгоритмом: він дорівнює сумі попарних добутків i -го рядка та j -го стовпця. Наприклад, другий елемент першого рядка обчислюється як сума попарних добутків відповідних елементів першого рядка і другого стовпця:

$$\begin{bmatrix} 1 & 2 & 1 \\ 3 & 4 & 0 \\ 1 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} 3 & 0 & 1 \\ -1 & 1 & 0 \\ 2 & 5 & 7 \end{bmatrix} = \\
 = \begin{bmatrix} 1 \cdot 3 + 2 \cdot (-1) + 1 \cdot 2 & 1 \cdot 0 + 2 \cdot 1 + 1 \cdot 5 & 1 \cdot 1 + 2 \cdot 0 + 1 \cdot 7 \\ 3 \cdot 3 + 4 \cdot (-1) + 0 \cdot 2 & 3 \cdot 0 + 4 \cdot 1 + 0 \cdot 5 & 3 \cdot 1 + 4 \cdot 0 + 0 \cdot 7 \\ 1 \cdot 3 + 0 \cdot (-1) + (-1) \cdot 2 & 1 \cdot 0 + 0 \cdot 1 + (-1) \cdot 5 & 1 \cdot 1 + 0 \cdot 0 + (-1) \cdot 7 \end{bmatrix} = \begin{bmatrix} 3 & 7 & 8 \\ 5 & 4 & 3 \\ 1 & -5 & -6 \end{bmatrix}$$

Текст програми з формою:

```

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{ const int n=3;   int i, j, k, a[n][n], b[n][n], c[n][n];
  for (i=0; i<n; i++)
  for (j=0; j<n; j++)
  { a[i][j]=Convert::ToInt32(dataGridView1[j,i]->Value); // Введення
    b[i][j]=Convert::ToInt32(dataGridView2[j,i]->Value); // матриць
  }
  for (i=0; i<n; i++) //Обчислення матриці-результату
  for (j=0; j<n; j++)
  { c[i][j]=0;
    for (k=0; k<n; k++) c[i][j]+=a[i][k]*b[k][j];
  }
}
  
```

```

//Виведення результату
for (i=0; i<n; i++)
for (j=0; j<n; j++) dataGridView3[j,i]->Value=c[i][j].ToString();
}
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{ dataGridView1->ColumnCount = 3; // Кількість стовпців
  dataGridView2->ColumnCount = 3;
  dataGridView3->ColumnCount = 3;
  dataGridView1->Rows->Add(3); // Кількість рядків
  dataGridView2->Rows->Add(3);
  dataGridView3->Rows->Add(3);
  // Прибрати заголовки стовпців
  dataGridView1->ColumnHeadersVisible = false;
  dataGridView2->ColumnHeadersVisible = false;
  dataGridView3->ColumnHeadersVisible = false;
  // Прибрати заголовки рядків
  dataGridView1->RowHeadersVisible = false;
  dataGridView2->RowHeadersVisible = false;
  dataGridView3->RowHeadersVisible = false;
  // Не додавати автоматично рядки
  dataGridView1->AllowUserToAddRows = false;
  dataGridView2->AllowUserToAddRows = false;
  dataGridView3->AllowUserToAddRows = false;
  // Зменшити ширину стовпців
  for(int i=0; i<3; i++)
  { dataGridView1->Columns[i]->Width = 40;
    dataGridView2->Columns[i]->Width = 40;
    dataGridView3->Columns[i]->Width = 40;
  }
}

```

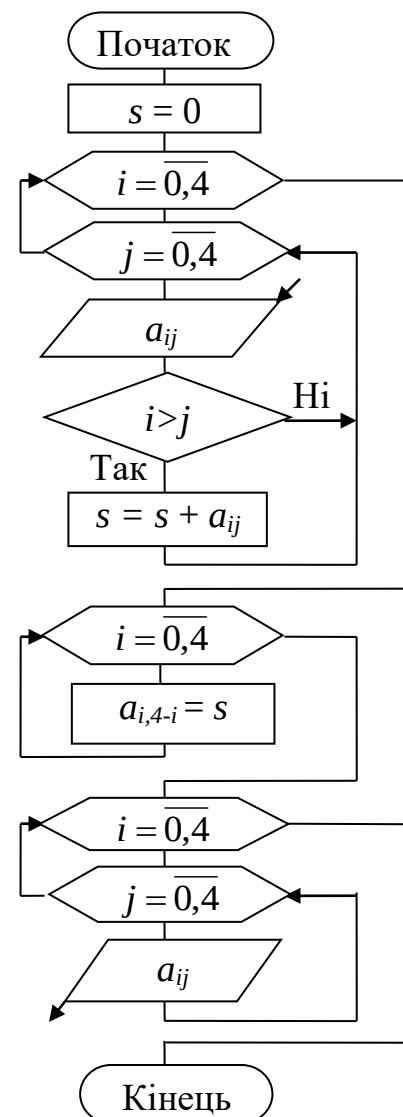
Приклад 5.27. Ввести матрицю цілих чисел розмірністю 5×5 та обчислити суму елементів в секторі під головною діагоналлю. Замінити елементи побічної діагоналі на цю суму.

Текст консольної програми
та блок-схема:

```

#include <iostream>
using namespace std;
int main()
{ const int n=5;
  system("color F0");
  setlocale(0, ".1251");
  int i, j, a[n][n], s=0;
  cout << "Введіть матрицю 5x5:" << endl;
  for (i=0; i<n; i++)
  for (j=0; j<n; j++) cin >> a[i][j];
}

```




```

// Сума елементів в секторі під головною діагоналлю
for (i=0; i<5; i++)
for (j=0; j<5; j++)
    if (i>j) s += a[i][j];
// Замінити елементи побічної діагоналі на цю суму
for (i=0; i<5; i++) a[i][4-i]=s;
cout<<"Матриця, в якій елементи побічної діагоналі \nзамінили сумою
    елементів \nпід головною діагоналлю:"<<endl;
for (i=0; i<n; i++)
{ for (j=0; j<n; j++) cout << a[i][j] << "\t";
  cout << endl;
}
system ("pause>>void");
return 0;
}

```

Результати роботи:

```

Введіть матрицю 5x5:
1      5      -6      12      9
0      66      2      0      -2
5      1      3      -5      11
2      3      1      7      2
-5     0      10     3      -1
Матриця, в якій елементи побічної діагоналі
замінили сумою елементів
під головною діагоналлю:
1      5      -6      12     20
0      66      2     20     -2
5      1     20     -5     11
2     20      1      7      2
20     0     10      3     -1

```

5.3.4 Опрацювання двовимірних масивів у функціях

При передаванні до функції *багатовимірних масивів* усі розмірності, якщо вони є невідомі на етапі компіляції, мають передаватися як параметри. Наприклад, заголовок функції, яка обчислює суму елементів динамічного двовимірного масиву, може мати вигляд:

```
int sum(int **a, int m, int n);
```

Виклик цієї функції у програмі може бути таким:

```
cout << "Сума елементів a = " << sum((int**)a,4,6);
```

Для звичайного статичного двовимірного масиву, коли обидві розмірності є відомі та є константами, заголовок функції матиме вигляд

```
int sum(int a[4][6]);
```

При опрацюванні у функціях як *одновимірних*, так і *двовимірних масивів*, які є параметрами цих функцій, насправді до функції передаються не самі масиви, а вказівники на їхні перші елементи.

Розглянемо кілька варіантів передавання матриці цілих чисел `a[3][4]` до функції `print()`, яка організовує виведення цієї матриці на екран у консолі.

1) Якщо розміри обох індексів є відомі, то проблем немає:

```
void print(int a[3][4])
{ for (int i=0; i<3; i++)
  { for (int j=0; j<4; j++) cout << a[i][j] << "\t";
    cout << endl;
  } }

```

Матриця i тут передається як вказівник, а розмірності наведено просто для повноти опису. Виклик такої функції може бути таким:

```
int x[3][4];
print(x);

```

2) Перша розмірність для обчислення адреси елемента є неважлива, тому її можна передавати як параметр:

```
void print(int a[][4], int m)
{ for (int i=0; i<m; i++)
  { for (int j=0; j<4; j++) cout << a[i][j] << "\t";
    cout << '\n';
  } }

```

Виклик цієї функції:

```
int x[3][4];
print(x, 3);

```

3) Найскладніший випадок – коли треба передавати обидві розмірності. Наведений нижче код функції є поширеною помилкою:

```
void print(int a[][[]], int m, int n)    // Помилка!
{ for (int i=0; i<m; i++)
  { for (int j=0; j<n; j++) cout << A[i][j] << "\t";
    cout << '\n';
  } }

```

По-перше, опис параметра $a[][]$ є неприпустимий, позаяк для обчислення адреси елемента двовимірного масиву слід знати другу розмірність. У такому разі найбільш поширеним і грамотним є застосування динамічних масивів, створювання яких більш докладно розглянуто у розд. 6. Наведемо правильний код такої функції:

```
void print(int **A, int m, int n)
{ for (int i=0; i<m; i++)
  { for (int j=0; j<n; j++) cout << ' ' << A[i][j];
    cout << '\n';
  } }

```

Виклик такої функції матиме вигляд:

```
int **x=new int*[3];
for (i=0; i<3; i++) x[i]=new int[4];
print(x,3,4);

```

Приклад 5.28. Ввести матрицю дійсних чисел розмірності 4×3 й обчислити за допомогою функції мінімальний елемент матриці та його індекси.

Розв'язок. Оскільки функція має обчислити і повернути три значення, то один з результатів – мінімальний елемент – буде повернено в головну програму через оператор `return`, а інші два результати – індекси – передаватимуться за посиланням (чи за вказівником).

Схеми алгоритму функції та основної програми:

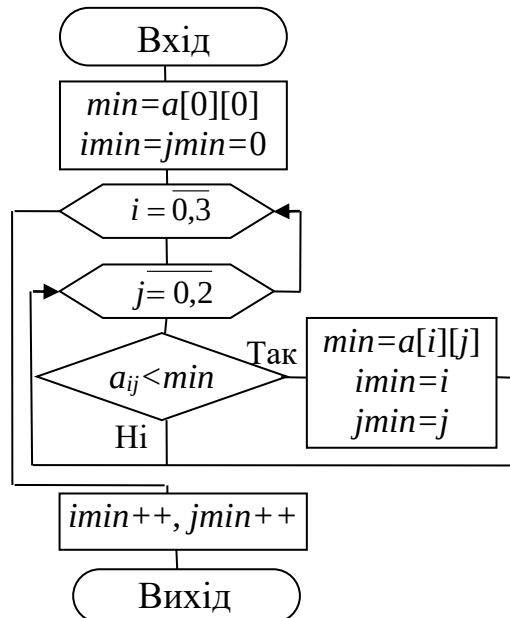


Схема алгоритму
функції **MIN ()**

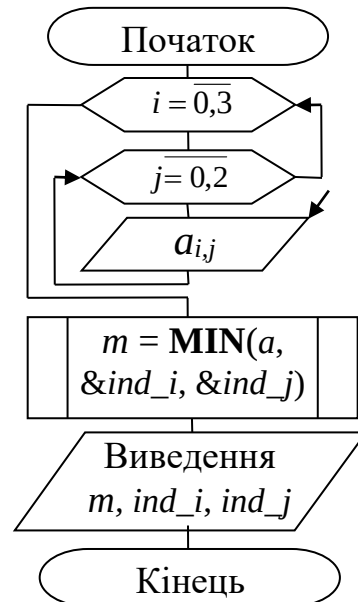


Схема алгоритму
до основної програми

Тексти функції та її виклику в основній програмі:

```

#include <iostream>
using namespace std;

double MIN(double a[4][3], int& imin, int& jmin)
{ double min=a[0][0];
  imin=jmin=0;
  for (int i=0; i<4; i++)
    for (int j=0; j<3; j++)
      if (a[i][j]<min)
        { min=a[i][j]; imin=i; jmin=j;
        }
  imin++; jmin++;
  return min;
}

int main ()
{ setlocale(0, ".1251");
  double a[4][3], m;   int i, j, ind_i, ind_j;
  cout<<" Введіть матрицю з 4-х рядків і 3-х стовпців:"<<endl;
  for (i=0; i<4; i++)
    for (j=0; j<3; j++) cin>>a[i][j];
  m = MIN(a, ind_i, ind_j);
}
  
```

```

cout<< "\n Мінімальний елемент " << m << "\n розміщено у "
    << ind_i << "-му рядку і " << ind_j << "-му стовпці" << endl;
system ("pause>>void");
return 0;
}

```

Результати роботи консольної програми:

Введіть матрицю з 4-х рядків і 3-х стовпців:

```

34      0      4.3
12.7    -7.1    0.3
-5       9      -1
2       -9.4    1.5

```

Мінімальний елемент -9.4

розміщено у 4-му рядку і 2-му стовпці

Приклад 5.29. Ввести матрицю дійсних чисел розмірності 5×5 і за допомогою функції замінити елементи головної діагоналі на середнє арифметичне відповідного рядка.

Схеми алгоритму функції та основної програми:

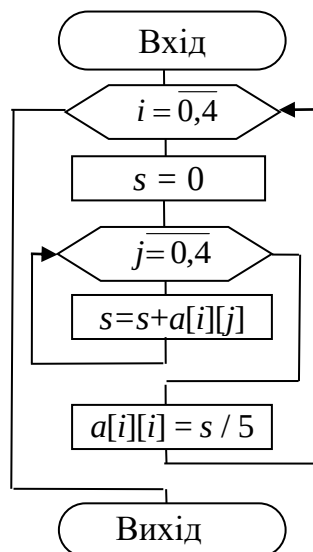


Схема алгоритму

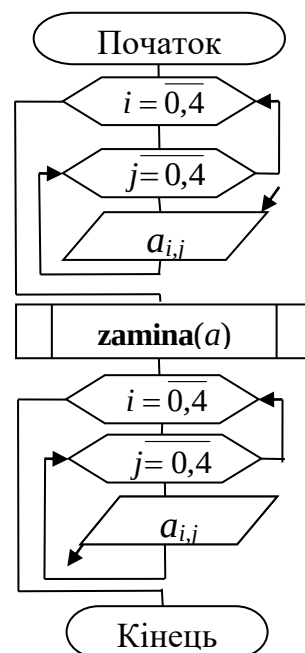


Схема алгоритму
до основної програми

Текст функції та основної програми:

```

#include <iostream>
using namespace std;

void zamina(double a[5][5])
{ for (int i=0; i<5; i++)
  { double s=0;
    for (int j=0; j<5; j++) s+=a[i][j]/5;
    a[i][i]=s;
  } }

```

```

int main ()
{ setlocale(0, ".1251");
  double a[5][5];
  int i, j;
  cout<<"Введіть матрицю з 5-ти рядків і 5-ти стовпців:"<<endl;
  for (i=0; i<5; i++)
  for (j=0; j<5; j++) cin >> a[i][j];
  zamina(a);
  cout<< "\nМатриця, в якій елементи головної діагоналі поміняли
          \на середнє арифметичне відповідного рядка:" << endl;
  for (i=0; i<5; i++)
  { for (j=0; j<5; j++) cout << a[i][j] << "\t";
    cout << endl;
  }
  system ("pause>>void");
  return 0;
}

```

Результати роботи консольної програми:

Введіть матрицю з 5-ти рядків і 5-ти стовпців:

23	6	0	-6.1	45.5
-6	0	1.5	4	-1
6.7	4	99	0	1
9	11	-5	5.8	0.1
3	3.2	7	49	9

Матриця, в якій елементи головної діагоналі поміняли
на середнє арифметичне відповідного рядка:

13.68	6	0	-6.1	45.5
-6	-0.3	1.5	4	-1
6.7	4	22.14	0	1
9	11	-5	4.18	0.1
3	3.2	7	49	14.24

Приклад 5.30. Розробити програму для введення матриці цілих чисел розмірністю 6×3 і створити функцію обчислення елементів вектора як середні арифметичні значення непарних елементів парних рядків.

Розв'язок. У функції `Vektor()` для обчислення елементів вектора спочатку слід організувати цикл для переміщення по парних рядках (2, 4, 6). У кожному з парних рядків, рухаючись по стовпцях, підсумовуємо тільки парні елементи та їхню кількість. Перед обчисленням середнього арифметичного, тобто перед діленням, перевіряємо можливість відсутності в даному парному рядку непарних елементів, тобто уніможливуємо поділ на нуль. Змінна k є потрібна для послідовного формування індексів вектора, позаяк їхня нумерація не збігається з нумерацією рядків у матриці.

Після розміщення на формі двох елементів `dataGridView` слід створити три стовпці для `dataGridView1` та один – для `dataGridView2`. Налаштування обох `dataGridView` організуємо у програмному коді для функції `Form1_Load`.

Текст функції та основної програми:

```

void Vektor(int a[6][3], double X[3])
{ int i, j, kol, k = 0;
  for(i=1; i<6; i+=2)
  { X[k] = kol = 0;
    for(j=0; j<3; j++)
      if(a[i][j] % 2 == 1)
        { X[k] += a[i][j]; kol++; }
    if (kol) X[k] /= kol ;    else X[k] = 0;
    k++;
  } }

private: System::Void button1_Click(System::Object^sender,System::EventArgs^e)
{ int i, j, a[6][3];    double V[3];
  for (i=0; i<6; i++)
  for (j=0; j<3; j++) a[i][j]=Convert::ToInt32(dataGridView1[j,i]->Value);
  Vektor(a, V);          // Виклик функції Vektor
  for(i=0; i<3; i++)
    dataGridView2[0,i]->Value =V[i].ToString("0.##");
}

private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{ dataGridView1->AllowUserToAddRows = false;
  dataGridView2->AllowUserToAddRows = false;
  for (int i=0; i<3; i++)
    dataGridView1->Columns[i]->DefaultCellStyle->Alignment =
      DataGridViewContentAlignment::MiddleCenter;
  dataGridView2->Columns[0]->DefaultCellStyle->Alignment =
    DataGridViewContentAlignment::MiddleCenter;
  dataGridView1->RowHeadersWidth = 90;
  dataGridView2->RowHeadersVisible = false;
  dataGridView1->Rows->Add(6);  dataGridView2->Rows->Add(3);
  for (int i=0; i<6; i++)
    dataGridView1->Rows[i]->HeaderCell->Value = Convert::ToString(i+1)+"-й рядок";
  for (int j=0; j<3; j++)
    dataGridView1->Columns[j]->HeaderText = Convert::ToString(j+1)+"-й стовпець";
  dataGridView2->Columns[0]->HeaderText = "Вектор";
  button2_Click(button2, e); // Виклик функції button2_Click
}

private: System::Void button2_Click(System::Object^sender,System::EventArgs^e)
{ Random^ rnd = gcnew Random();
  for (int i=0;i<6;i++)
    for (int j=0;j<3;j++) dataGridView1->Rows[i]->Cells[j]->Value=rnd->Next(50);
}

private: System::Void button3_Click(System::Object^sender,System::EventArgs^e)
{ for (int i=0; i<6; i++)
  for (int j=0; j<3; j++) dataGridView1[j,i]->Value = "";
  for (int i=0; i<3; i++) dataGridView2[0,i]->Value = "";
}

private: System::Void button4_Click(System::Object^ sender,System::EventArgs^e)
{ Close(); }

```

Результати роботи:

Опрацювання двовимірних масивів у функціях

	1-й стовпець	2-й стовпець	3-й стовпець
1-й рядок	33	28	40
2-й рядок	29	30	37
3-й рядок	8	34	5
4-й рядок	30	48	34
5-й рядок	37	30	38
6-й рядок	43	0	11

Сформувати вектор як середні арифметичні значення непарних елементів парних

Вектор

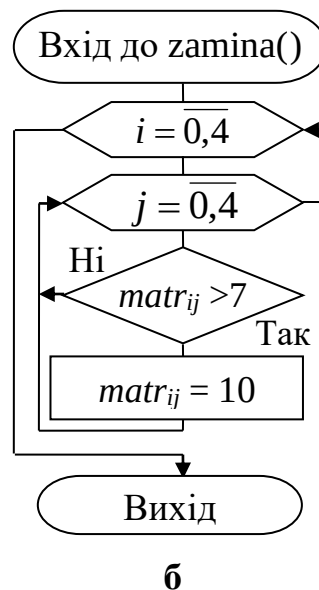
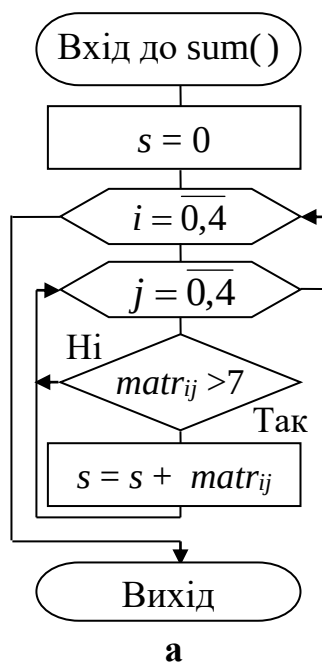
33
0
27

Очистити

Вихід

Заповнити матрицю випадковими елементами

Приклад 5.31. Розробити програму для введення матриці розмірністю 5×5 цілих чисел, обчислення суми елементів зі значенням понад 7 і замінити у вихідній матриці ці елементи на значення 10 та обчислити нову суму елементів матриці зі значенням понад 7.

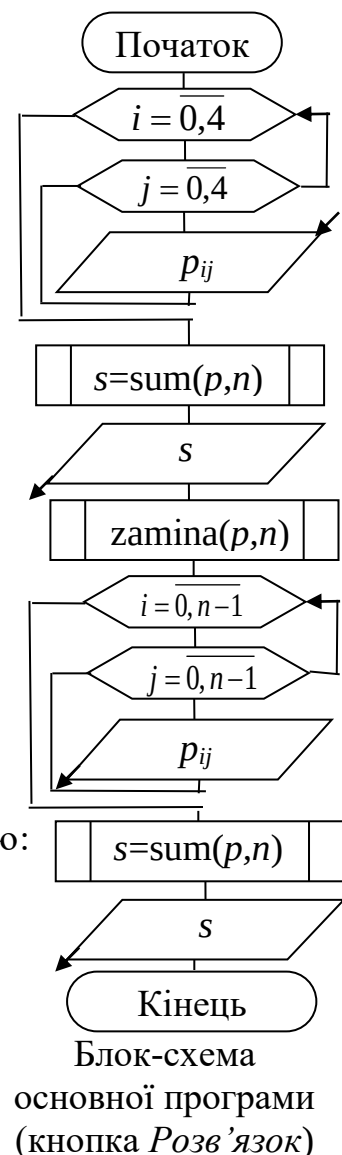


Блок-схеми функцій:
а – `sum()`; б – `zamina()`

Тексти функцій та основної програми з формою:

```

int sum(int matr[5][5])
{
    int i, j, s=0;
    for (i=0; i<5; i++)
        for (j=0; j<5; j++)
            if (matr[i][j] > 7) s += matr[i][j];
    return s;
}
  
```



```
void zamina(int matr[5][5])
{
    for (int i=0; i<5; i++)
        for (int j=0; j<5; j++)
            if (matr[i][j] > 7)
                matr[i][j] = 10;
}

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    const int n=5;
    int i, j, s, a[n][n];
    for (i=0; i<n; i++)
        for (j=0; j<n; j++) a[i][j]=Convert::ToInt32(dataGridView1[j,i]->Value);
    s=sum(a);
    textBox1->Text=s.ToString();
    zamina(a);
    for (i=0; i<5; i++)
        for (j=0; j<5; j++) dataGridView2[j,i]->Value=a[i][j].ToString();
    s=sum(a);
    textBox2->Text=s.ToString();
    for (i=0; i<n; i++)
        for (j=0; j<n; j++) dataGridView2[j,i]->Value=a[i][j].ToString();
}

private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
    // Кількість стовпців
    dataGridView1->ColumnCount = 5;
    dataGridView2->ColumnCount = 5;
    // Кількість рядків
    dataGridView1->Rows->Add(5);
    dataGridView2->Rows->Add(5);
    // Прибрати заголовки стовпців
    dataGridView1->ColumnHeadersVisible = false;
    dataGridView2->ColumnHeadersVisible = false;
    // Прибрати заголовки рядків
    dataGridView1->RowHeadersVisible = false;
    dataGridView2->RowHeadersVisible = false;
    // Не додавати автоматично рядки
    dataGridView1->AllowUserToAddRows = false;
    dataGridView2->AllowUserToAddRows = false;
    // Зменшити ширину стовпців
    for(int i=0; i<5; i++)
    {
        dataGridView1->Columns[i]->Width = 40;
        dataGridView2->Columns[i]->Width = 40;
    }
}
```


Ввести матрицю 5x5 цілих чисел

11	2	-2	-56	102
9	4	3	0	-3
-65	3	45	1	6
1	23	7	10	50
4	2	-3	1	-2

Сума елементів понад 7 250

Матриця, в якій елементи понад 7 замінено на 10

10	2	-2	-56	10
10	4	3	0	-3
-65	3	10	1	6
1	10	7	10	10
4	2	-3	1	-2

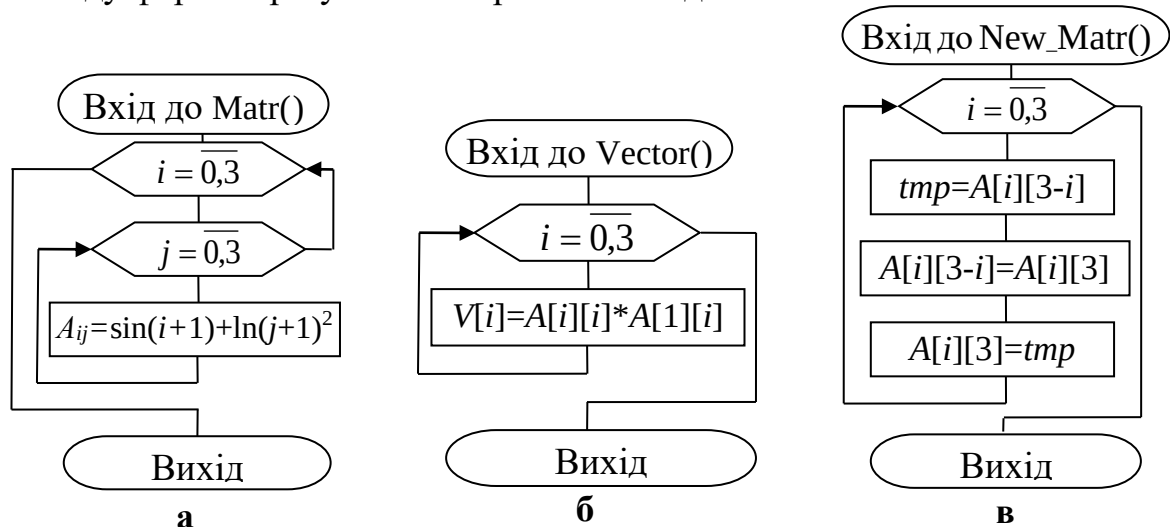
Нова сума елементів понад 7 70

Розв'язок

Приклад 5.32. Розробити схему алгоритму та програмний проект для:

- ✓ обчислення елементів матриці розмірністю 4×4 за формулою $A_{ij} = \sin(i) + \ln(j^2)$, де $i = \overline{0,4}$, $j = \overline{0,4}$;
- ✓ обчислення елементів вектора добутків елементів головної діагоналі матриці на елементи другого рядка;
- ✓ переставляння елементів побічної діагоналі й останнього стовпця.

Розв'язок. Схеми алгоритмів функцій та основної програми, а також приклад вигляду форми з результатами роботи наведено нижче.



Блок-схеми функцій:

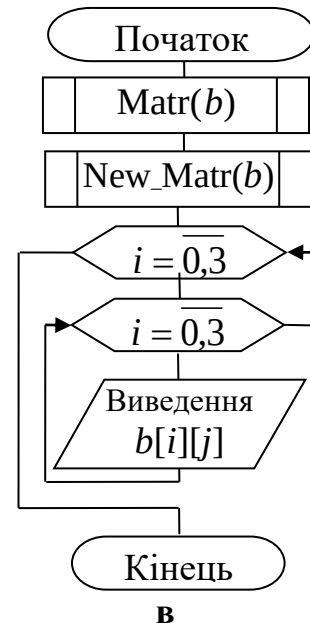
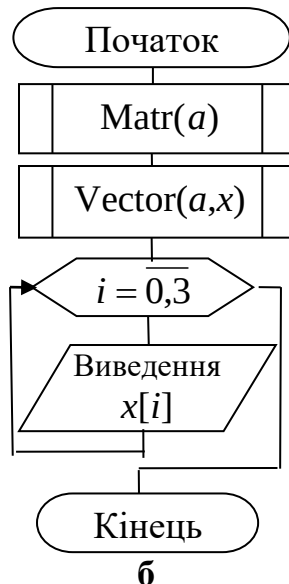
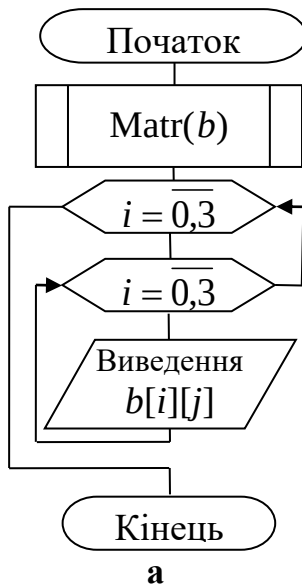
а – **Matr()**; б – **Vector()**; в – **New_Matr()**

Тексти функцій:

```

void Matr( double A[4][4]) // Функція обчислення елементів матриці
{ for (int i=0; i<4; i++)
  for (int j=0; j<4; j++) A[i][j]=Math::Sin(i+1)+Math::Log((j+1)*(j+1));
}
// Функція обчислення елементів вектора
void Vector(double A[4][4], double V[])
{ for (int i=0; i<4; i++)
  V[i]=A[i][i]*A[1][i];
}
  
```

```
// Функція переставлення елементів неголовної
// діагоналі й останнього стовпчика матриці
void New_Matr(double A[4][4])
{ double tmp;
  for (int i=0; i<4; i++)
  { tmp = A[i][3-i];
    A[i][3-i] = A[i][3];
    A[i][3] = tmp;
  }
}
```



Блок-схеми для кнопок:
а – button1; б – button2; в – button3

Тексти програм для командних кнопок:

```
private: System::Void button1_Click
(System::Object^ sender, System::EventArgs^ e)
{ const int n=4;
  double b[n][n];
  Matr(b); // Виклик функції
  for (int i=0; i<n; i++)
  for (int j=0; j<n; j++)
    dataGridView1[j,i]->Value =
      b[i][j].ToString("0.###");
}
private: System::Void button2_Click
(System::Object^ sender, System::EventArgs^ e)
{ double a[4][4], x[4];
  Matr(a); // Виклик функції
  Vector(a,x); // Виклик функції
  for (int j=0; j<4; j++)
    dataGridView2[j,0]->Value=x[j].ToString("0.###");
}
```

Обчислити матрицю			
0,841	2,228	3,039	3,614
0,909	2,296	3,107	3,682
0,141	1,527	2,338	2,914
-0,757	0,629	1,44	2,016

Вектор добутків елементів головної діагоналі матриці на елементи другого рядка			
0,765	5,27	7,264	7,422

Переставити елементи побічної діагоналі й останнього стовпця			
0,841	2,228	3,039	3,614
0,909	2,296	3,682	3,107
0,141	2,914	2,338	1,527
2,016	0,629	1,44	-0,757

```

private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e)
{ double b[4][4];
  Matr(b);      // Виклик функції
  New_Matr(b);  // Виклик функції
  for (int i=0; i<4; i++)
  for (int j=0; j<4; j++) dataGridView3[j,i]->Value=b[i][j].ToString("0.###");
}
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
  dataGridView1->ColumnCount = 4; // Кількість стовпців
  dataGridView2->ColumnCount = 4;
  dataGridView3->ColumnCount = 4;
  dataGridView1->Rows->Add(4);    // Кількість рядків
  dataGridView2->Rows->Add(1);
  dataGridView3->Rows->Add(4);
  // Прибрати заголовки стовпців
  dataGridView1->ColumnHeadersVisible = false;
  dataGridView2->ColumnHeadersVisible = false;
  dataGridView3->ColumnHeadersVisible = false;
  // Прибрати заголовки рядків
  dataGridView1->RowHeadersVisible = false;
  dataGridView2->RowHeadersVisible = false;
  dataGridView3->RowHeadersVisible = false;
  // Не додавати автоматично рядки
  dataGridView1->AllowUserToAddRows = false;
  dataGridView2->AllowUserToAddRows = false;
  dataGridView3->AllowUserToAddRows = false;
  // Зменшити ширину стовпців
  for(int i=0; i<4; i++)
  { dataGridView1->Columns[i]->Width = 50;
    dataGridView2->Columns[i]->Width = 50;
    dataGridView3->Columns[i]->Width = 50;
  }
}

```

Питання та завдання для самоконтролю

- 1) Дати означення масиву у програмуванні?
- 2) В який спосіб задають розмірність масиву?
- 3) Яке призначення індексів масиву? Змінні яких типів можна використовувати для індексів? В який спосіб записують індекси масиву?
- 4) Які з наведених оголошень одновимірного масиву з 10-ти елементів помилкові і чому?

а) <code>int A[1..10];</code>	в) <code>int A[9];</code>
б) <code>double A[10];</code>	г) <code>double A[0, 9];</code>
- 5) Записати оголошення одновимірного масиву з 25-ти дійсних чисел.
- 6) Навести оператор, який присвоює першому елементу масиву А з 20-ти цілих чисел нульове значення.
- 7) Записати оператор оголошення символьного масиву S з 10-ти елементів.

8) Записати оголошення одновимірного констант-масиву з послідовності символів Вашого прізвища.

9) Які елементи керування Visual C++ для введення елементів одновимірних масивів на форму Ви знаєте?

10) Записати оператори виведення масиву W з 17-ти дійсних чисел до трьох різних елементів керування.

11) Вибрати правильні твердження:

- а) якщо список ініціалізації містить початкових значень менше за розмірність масиву, це є помилка;
- б) оголошення `int A[23];` резервує пам'ять під масив A з 23 елементів цілого типу;
- в) оголошення `float A = {-1, 2.2, 5};` резервує пам'ять для трьох елементів масиву A типу `float`;
- г) масив може зберігати дані різних типів.

12) В яких випадках при оголошенні масиву можна не задавати його розмірність?

13) Найти і розтлумачити помилки у твердженнях:

- а) індекси масиву можуть бути якого завгодно типу;
- б) при оголошенні масиву програма автоматично ініціалізує його нульовими значеннями;
- в) щоб звернутися до конкретного елемента масиву, слід записати ім'я масиву і значення цього елемента;
- г) кількість елементів масиву завжди збігається з кількістю займаних ним байтів оперативної пам'яті.

14) Вибрати ім'я змінної, значенням якої є сума елементів масиву в наведених трьох фрагментах програм:

а) <code>s1=0;</code>	б) <code>s2=1;</code>	в) <code>s3=0;</code>
<code>for(i=0;i<10;i++)</code>	<code>for(i=0;i<10;i++)</code>	<code>for(i=0;i<10;i++)</code>
<code>s1 += a[i];</code>	<code>s2 *= a[i];</code>	<code>if(a[i]>0) s3++;</code>

15) Записати ім'я змінної, значенням якої є кількість додатних елементів масиву в наведених трьох фрагментах програм:

а) <code>S1=0;</code>	б) <code>S2=1;</code>	в) <code>S3=0;</code>
<code>for(i=0;i<10;i++)</code>	<code>for(i=0;i<10;i++)</code>	<code>for(i=0;i<10;i++)</code>
<code>S1 += a[i];</code>	<code>S2 *= a[i];</code>	<code>if(a[i]>0) S3++;</code>

16) Записати ім'я змінної, значенням якої є максимальний з елементів масиву в наведених трьох фрагментах програм:

а) <code>S1=0;</code>	б) <code>S2=a[0];</code>	в) <code>S3=0;</code>
<code>for(i=0;i<10;i++)</code>	<code>for(i=1;i<10;i++)</code>	<code>for(i=0;i<10;i++)</code>
<code>S1 += a[i];</code>	<code>if(a[i]>S2)S2=a[i];</code>	<code>if(a[i]>0)S3=S3+1;</code>

17) Які з наведених оголошень двовимірних масивів неправильні й чому?

- а) `int C[1..5, 1..5];`
- б) `float C[5][5];`
- в) `float C[1..5][1..5];`
- г) `int C: [5][5];`

18) В який спосіб розміщуються елементи двовимірних масивів в оперативній пам'яті?

19) Записати фрагмент програми для введення з елемента керування `dataGridView1` елементів матриці W з 3-х рядків і 5-ти стовпців цілих чисел.

20) Задати при оголошенні матриці цілих чисел z з 3-х рядків і 2-х стовпців початкові значення її елементам.

21) Записати оператор оголошення символьної матриці S розміром 7×3 .

22) Під яким номером записано оператор, що обчислює суму елементів головної діагоналі матриці A цілих чисел розміром 5×5 ?

- а) `for(i=0, s=0; i<5; i++) s++;`
- б) `for(i=0, s=0; i<5; i++) s+=A[i][i];`
- в) `for(i=0, s=0; i<5; i++) for(j=0; j<5; j++) s+=A[i][j];`
- г) `for(i=0, s=0; i<5; i++) A[i][i]=0;`

23) Записати фрагмент програми обчислення кількості елементів масиву A з 10-ти цілих чисел, значеннями яких є двоцифрові числа.

24) Записати фрагмент програми для обчислення середнього арифметичного, максимального і мінімального елементів масиву Z з 15-ти цілих чисел.

25) Вибрати правильні оголошення масивів:

- а) `mas[7]={1,3,0,-2,0,1,9};`
- б) `char s[10];`
- в) `const int m=9; int A[m];`
- г) `float C[5]={12,4,82};`
- д) `int B[]={1,20,3,0,-1};`
- е) `int w[5]={-1,2,40,8,20,9};`

26) Вибрати номер правильного оператора виведення на екран у консольному режимі масиву mas з 7-ми цілих чисел:

- а) `cout << mas;`
- б) `for(int i=0; i<7; i++) cout << mas[i];`
- в) `for(int i=0; i<=7; i++) cout << mas[i];`
- г) `for(int i=1; i<7; i++) cout << mas[i];`
- д) `for(int i=1; i<=7; i++) cout << mas[i];`

27) Віднайти і розтлумачити помилки у фрагментах програм для `int s,i,a[10];`

- а) `for(s=0, i=0; i<10; i++) { cin >> a[i]; s += a[i];}
cout << "Середнє арифметичне введенних чисел =" << s/10;`
- б) `for(i=0; i<10; i++) { cin >> a[i]; s += a[i];}
cout << "Сума елементів масиву =" << s;`
- в) `for(s=0, i=0; i<5; i+=2) { cin >> a[i]; s += a[i];}
cout << "Сума непарних елементів масиву =" << s/5.;`
- г) `for(s=0, i=0; i<5; i++) { cin >> a[i]; s *= a[i];}
cout << "Добуток елементів масиву =" << s;`

27) Записати фрагмент програми обчислення остачі від ділення максимального на мінімальний елемент масиву з 10-ти цілих чисел.

28) Записати фрагмент програми упорядкування за спаданням масиву з 20-ти цілих чисел.

29) Чи можна виконувати опрацювання двовимірної масиву, організувавши зовнішній цикл по стовпцях, а внутрішній – по рядках?

30) Чи варто організовувати вкладені цикли для опрацювання лише головної діагоналі квадратної матриці?

РОЗДІЛ 6

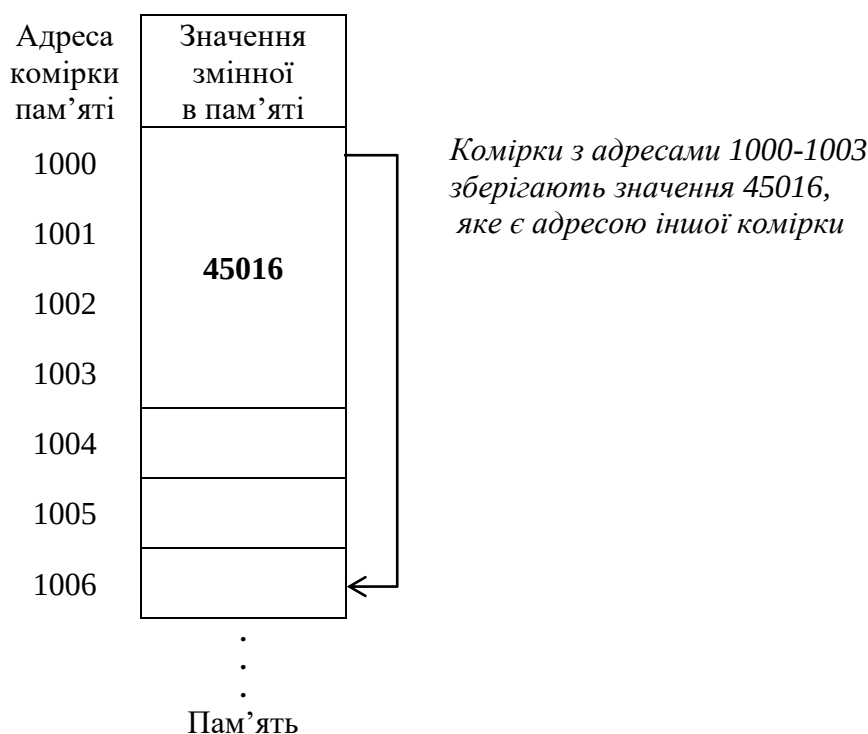
ВКАЗІВНИКИ. ДИНАМІЧНА ПАМ'ЯТЬ

6.1 Вказівники

Пам'ять комп'ютера являє собою сукупність комірок для зберігання інформації, кожна з яких має власний номер. Ці номери називаються *адресами*. Розмір однієї комірки пам'яті становить 1 байт. При оголошенні змінної в пам'яті виділяється ділянка обсягом, відповідним до типу змінної, наприклад, 4 байти для типу `int`. Ця ділянка пам'яті пов'язується з іменем змінної. Можна сказати, що адреса змінної – це адреса першої комірки цієї ділянки. Існують спеціальні змінні – *вказівники*, в яких можна зберігати адреси комірок пам'яті, тобто адреси змінних.

Вказівник – це змінна, яка містить адресу іншої змінної, при чому вказівник вказує на змінну того типу, адресу якої він містить.

Слід звернути увагу на те, що *адреса* і *значення* змінної – це зовсім різні поняття. Зазвичай адреси змінних не є важливими, найголовніше – значення цих змінних. Вказівники використовуються для прямого доступу до даних у пам'яті.



Вказівник оголошується за допомогою *:

`<тип> *<ім'я>;`

Тут *тип* – це базовий тип вказівника, котрим може бути який завгодно тип. *Ім'я* є ідентифікатором змінної-вказівника. Слід звернути увагу на те, що *тип* – це не тип вказівника, а тип даних, адресу яких буде записано у вказівнику.

Наприклад, вказівник на пам'ять, в якій зберігатиметься ціле число:

```
int *A;
```

Вказівник на дійсне число:

```
float *B;
```

При цьому змінна *A* – це адреса ділянки пам'яті, в якій розташоване ціле число, *B* – адреса ділянки пам'яті, в якій розташоване дійсне число. Обидві змінні – *A* та *B* – є вказівниками, тобто їхніми значеннями є адреси. Окрім того, адреси змінних типу *int* та *float* займають у пам'яті однакову кількість байтів. Але насправді змінні *A* та *B* мають різний тип: *A* – вказівник на ціле, *B* – вказівник на дійсне.

Базовий тип вказівника визначає тип об'єкта, адресу якого містить вказівник. Фактично вказівник будь-якого типу може посилатися на яке завгодно місце в пам'яті. Однак операції, які можуть виконуватися з вказівником, суттєво залежать від його типу. Наприклад, якщо оголошено вказівник типу *int **, компілятор припускає, що адреса, на яку він посилається, містить змінну типу *int*, хоча це може бути і не так. Отже, при оголошенні вказівника слід переконатися, що його тип є сумісним з типом об'єкта, на який він буде посилатися.

Вказівникові можна присвоїти лише адресу змінної відповідного типу, оскільки C++ не виконує автоматичного перетворення типів вказівників. Тому такі команди є помилковими:

```
A = B;    // Не можна переприсвоювати один одному вказівники на різні типи;  
A = 10;   // не можна присвоювати числа вказівникам.
```

У мові C++ визначено дві операції для роботи з вказівниками:

1) **&** – **адресація**, тобто отримання адреси змінної. Приміром, щоб отримати адресу змінної *x*, оголошеної як *int x=10*, можна скористатись командою *A = &x*;

2) ***** – **розадресація** чи розіменування вказівника. Щоб взяти значення *10*, з адреси у вказівнику *A* слід звернутися **A*, а щоб змінити це значення, збільшивши його на 1, можна скористатись командою *(*A)++* (при чому команда **A++* дасть помилкове рішення).

Звернімо увагу на те, що «зірочки» при оголошенні вказівника та операції розадресації – це зовсім різні речі, які лише позначаються однаковим символом.

Ініціалізувати вказівник треба до його використання, тобто присвоїти йому адресу якоїсь змінної або об'єкта. Вказівник, який не вказує на жодне значення, називається порожнім, чи нульовим. Такий вказівник має значення 0 чи NULL

Для прикладу у програмі оголосимо цілу змінну *X* зі значенням 7:

```
int X = 7;
```

тоді, щоб її адресу записати до вказівника *A*, слід записати команду:

```
A = &X;    // Значенням змінної A є адреса змінної X.
```

Якщо пам'ять, у якій зберігається змінна *X*, починається з комірки під номером 2000, тоді змінній *A* після цієї команди буде присвоєно значення 2000.

Тепер до значення 7, яке зберігається у змінній *X*, можна звернутися не лише за ім'ям *X*, але й через вказівник. При цьому слід застосувати операцію *розадресації* вказівника (вона позначається символом «*»), наприклад: **A*. Цей

запис означає: «значення, яке зберігається за адресою, записаною у змінній А». Щоб збільшити на 2 оголошену раніше змінну X, можна записати так:

```
X = X + 2;  /* За ім'ям X комп'ютер визначить адресу ділянки пам'яті, в
            якій вона розташована, візьме значення, що зберігається
            у пам'яті за цією адресою, збільшить його на 2 і запише
            назад у ту ж саму ділянку пам'яті */
```

чи так:

```
*A = (*A) + 2; /* Комп'ютер з ділянки пам'яті за адресою, записаною у
                змінній A, візьме значення, яке зберігається в ній, збі-
                льшить це значення на 2 і запише в ту ж саму ділянку */
```

У першому разі відбувається звертання до змінної з ім'ям X, у другому – до змінної через її адресу, яку записано у змінній-вказівнику A.

Звернімо увагу на те, що «зірочки» при оголошенні вказівника та операції розадресації – це зовсім різні речі, які лише позначаються однаковим символом.

Вказівник, який не вказує на жодне значення, називається порожнім, чи нульовим. Такий вказівник має значення 0 чи NULL.

Вказівники використовуються при роботі з динамічними змінними і при передаванні параметрів до функцій (див. розд. 8).

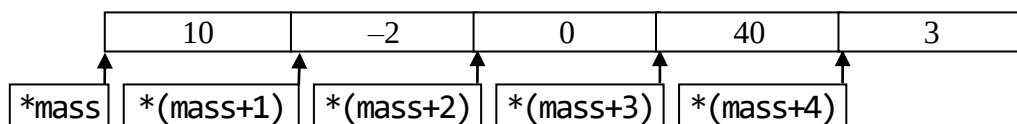
6.2 Вказівники на одновимірні масиви

Масиви і вказівники у C++ тісно пов'язані і можуть використовуватись майже еквівалентно. Ім'я масиву можна сприймати як константний вказівник на перший елемент масиву. Його відмінність від звичайного вказівника полягає у тому, що його не можна модифікувати.

Здійсимо оголошення масиву `mass` з п'яти цілих чисел з ініціалізацією значень елементів і вказівника на ціле `ptr`:

```
int mass[5] = {10, -2, 0, 40, 3}, *ptr;
```

При такому оголошенні масиву пам'ять виділяється не лише для п'яти елементів масиву, а й для вказівника з ім'ям `mass`, значення якого дорівнює адресі першого елемента масиву `mass[0]`, тобто сам масив залишається безіменним, а доступ до елементів масиву здійснюється через вказівник з ім'ям `mass`.



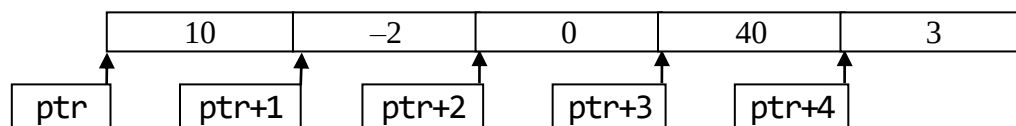
Для того, щоби звернутися до 3-го елемента цього масиву, можна записати чи то `mass[2]` чи `*(mass+2)`. При реалізації на комп'ютері перший з цих способів зводиться до другого, тобто індексний вираз перетворюється до адресного. Оскільки операції над вказівниками виконуються швидше, тому, якщо елементи масиву обробляються по чергові, то доцільніше використовувати другий спосіб. Якщо ж вибір елементів є випадковим, то, щоб уникнути помилок, прийнятнішим є перший спосіб. Крім того, перший спосіб є більш наочним і звичним для сприйняття, що сприяє кращій читабельності програм.

Оскільки ім'я масиву є вказівником на перший елемент масиву, можна надати вказівнику адресу першого елемента масиву за допомогою оператора:

```
ptr = mass;
```

Цей запис є еквівалентним присвоюванню адреси першого елемента масиву (тобто до елемента з нульовим індексом) у вигляді оператора

```
ptr = &mass[0];
```



Після цього звернутися до першого елемента масиву і надати йому значення 2 можна будь-яким із шести операторів:

```
* mass = 2;      mass[0] = 2;      *(mass+0) = 2;
* ptr = 2;        ptr[0] = 2;        *(ptr+0) = 2;
```

Усі ці оператори за дією є тотожними, але швидше за всі виконуватимуться присвоювання `*mass = 2` та `*ptr = 2`, оскільки в них не треба виконувати операції додавання.

Вказівники можна індексувати так само, як і масиви. Наприклад, вираз `ptr[3]` посилається до четвертого елемента масиву `mass[3]`.

Зауважимо, що не слід плутати такі оголошення:

```
int *p1[10];           // приклад 1
int (*p2)[10];         // приклад 2
```

У першому прикладі оголошено масив вказівників з ім'ям `p1`. Масив складається з 10 елементів, кожен з яких є вказівником на змінну типу `int`.

У другому прикладі оголошено змінну-вказівник з ім'ям `p2`, яка вказує на масив з 10 цілих чисел типу `int`.

6.3 Арифметика вказівників

Вказівники можуть застосовуватися як операнди в арифметичних виразах, виразах присвоювання і виразах порівняння. Проте не усі операції, зазвичай використовувані у таких виразах, дозволені для вказівників.

З вказівниками може виконуватися обмежена кількість арифметичних операцій. Вказівник можна збільшувати (`++`), зменшувати (`--`), додавати до вказівника цілі числа (`+` чи `+=`), віднімати від нього цілі числа (`-` чи `-=`) або віднімати один вказівник від іншого.

Здебільшого арифметичні операції застосовуються до вказівників на масиви. Більше того, арифметика вказівників втрачає всякий сенс, якщо вона виконується не над вказівниками на масив. Змінення (збільшення або зменшення) значення вказівника на масив по суті є зсувом адреси у межах цього масиву. Перетворення цілої величини до адресного зсуву припускає, що у межах зсуву щільно розташовані елементи однакового розміру. Це припущення справедливо для елементів масиву, оскільки масив визначається як набір величин однакового типу, а його елементи розташовані у суміжних комірках пам'яті. Додавання і

віднімання адрес, які посилаються на будь-які величини, крім елементів масиву, дає непередбачуваний результат, оскільки в такому разі не гарантується щільне заповнення пам'яті.

Розглянемо на прикладах дію арифметичних операцій на вказівники:

✓ *Додавання й віднімання цілого числа до вказівника* відрізняється від звичайної арифметики. При їх виконанні адреса, яка зберігається у вказівникові, змінюється кратно до розміру даних, на які вказує вказівник. Для оголошеного вище масиву `mass` і вказівника `ptr`, який вказує на початок масиву, команда

```
ptr += 3;
```

збільшить адресу, яка міститься у `ptr`, на $4 \cdot 3 = 12$ байти, і `ptr` вказуватиме на четвертий елемент масиву зі значенням 40.

Загальна формула, яка обчислює значення вказівника при додаванні (відніманні) цілого числа N :

$\langle \text{Нова адреса} \rangle = \langle \text{стара адреса} \rangle + / - \langle \text{розмір базового типу} \rangle * N$;

✓ *Операції інкременту (++) і декременту (--)*, тобто збільшення чи зменшення вказівника на 1. При цьому адреса, яка зберігається у цьому вказівнику, збільшується на розмір базового типу в байтах. Наприклад, команда

```
ptr++;
```

збільшить його значення на 4, оскільки розмір змінної типу `int` зазвичай дорівнює 4, а сам вказівник вказуватиме на наступний елемент масиву.

Зменшення вказівника на 1 означає віднімання розміру в байтах базового типу від адреси, яка зберігається у вказівнику.

✓ *Різниця однотипних вказівників*. Вказівники можна віднімати один від одного. Наприклад, якщо `ptr1` вказує на перший елемент масиву `mass`, а вказівник `ptr2` – на четвертий, то результат виразу `ptr2 - ptr1` має тип `int` і дорівнює 3 – різниці індексів елементів, на які вказують ці вказівники. І так буде, не дивлячись на те, що адреси, які містяться у цих вказівниках, розрізняються на 12 байтів (якщо елемент масиву займає 4 байти). Тобто, результат такої операції дорівнює кількості елементів вихідного типу між зменшуваним і від'ємником, причому якщо перша адреса молодше, то результат має від'ємне значення.

Загальна формула при відніманні однотипних вказівників:

$\langle \text{Кількість елементів} \rangle = (\langle \text{перший вказівник} \rangle - \langle \text{другий вказівник} \rangle) / \langle \text{розмір типу} \rangle$

Наприклад:

```
int *ptr1, *ptr2, mass[5]={10,-2,0,40,3}, i;
ptr1 = a + 4;
ptr2 = a + 9;
i = ptr1 - ptr2; /* дорівнює -5 */
i = ptr2 - ptr1; /* дорівнює 5 */
```

✓ *При порівнянні вказівників* порівнюються адреси, які містяться у вказівниках, а результат порівняння дорівнює 0 (**false**) або 1 (**true**). Порівняння вказівників операціями `>`, `<`, `>=`, `<=` мають сенс лише для вказівників на один той самий масив. Проте, операції відношення `==` та `!=` мають сенс для будь-яких

вказівників. При цьому вказівники є рівнозначними, якщо вони вказують на одну і ту саму адресу в пам'яті. Наприклад, для розглянутих вище вказівників ptr1 та ptr2 у команді

```
if (ptr1 > ptr2) mass[3]=4;
```

значення ptr1 є менше за значення ptr2 і тому оператор mass[3]=4 не буде виконаний.

Приклад порівняння вказівника ptr з ненульовим значенням:

```
if (ptr != NULL) MessageBox::Show("Ненульовий вказівник");
```

Типи вказівників у Visual C++

У Visual Studio, крім узагальненого і вже розглянутого нами базового поняття нерегульованих вказівників, розглядаються і регульовані вказівники.

Так, режим CLR вміє працювати не лише з «рідними» нерегульованими вказівниками, які позначаються *, і пам'ять для яких слід не лише виділяти в так званій некерованій купі, але й звільняти, інакше купа може переповнитись, і процес виконання програмного проекту перерветься. Режим CLR працює і з регульованими вказівниками, керування пам'яттю для яких відбувається під керуванням самого середовища. Для таких вказівників використовують спеціальну позначку: замість символу * застосовують символ ^. Регульовані вказівники повсюдно використовуються при роботі з елементами керування на формі. Так у середовищі VC++ існує спеціальна утиліта gsnw, яка формує екземпляр відповідного об'єкта, виділяючи йому пам'ять і повертаючи посилання (адресу) на цей екземпляр.

У зв'язку з виникненням більш «просунутих» регульованих вказівників виникла потреба апарата перетворення нерегульованих вказівників до регульованих і, навпаки. Цей процес назвали маршалізацією, а для забезпечення цього процесу створили спеціальну бібліотеку.

Доцільність використання вказівників

Правильне розуміння і використання вказівників має велике значення при створенні більшості C++-програм з чотирьох причин:

- 1) використання вказівників може підвищити ефективність роботи деяких функцій;
- 2) вказівники надають можливість «бачити» в основній програмі можливі модифікації аргументів у функціях;
- 3) вказівники використовуються для підтримки системи динамічного виділення пам'яті;
- 4) вказівники використовуються для підтримки певних структур даних, а саме: зв'язані списки і бінарні дерева.

Крім того, що вказівники – одна з найсильніших сторін C, вони, в той самий час, можуть завдати великої шкоди. Наприклад, неініціалізований чи дикий вказівник може спричинити крах системи, може бути навіть гірше, коли некоректне використання вказівників призводить до невловимих помилок.

Приклад 6.1 В одновимірному масиві з 8 цілих чисел обчислити кількість додатних та суму парних елементів.

Розв'язок. Для розв'язання організуємо функцію, параметрами якої буде вказівник на масив, розмірність масиву та адреса одного з обчислюваних результатів. Інший результат передаватимемо з функції через оператор `return`. Звернення `*(a+i)` є аналогічним до `a[i]` і тому можна однаково використовувати як те, так і те.

Текст функції та основної програми:

```
#include <iostream>
#include <conio.h>
using namespace std;
int fun(int *a, int n, int &s)
{ int k=0; s=0;
  for (int i=0; i<n; i++)
  { if (*(a+i) > 0) k++;
    if (*(a+i)%2 == 0) s += *(a+i);
  }
  return k;
}
int main()
{
  setlocale(0, ".1251");
  system("color F0");
  int i, a[8], sum, kol;
  cout << "Введіть 8 цілих чисел:\n";
  for(i=0; i<8; i++)
    cin >> *(a+i);
  kol = fun(a,8,sum);
  cout << "Кількість додатних елементів = " << kol << endl;
  cout << "Сума парних елементів = " << sum << endl;
  _getch();
  return 0;
}
```

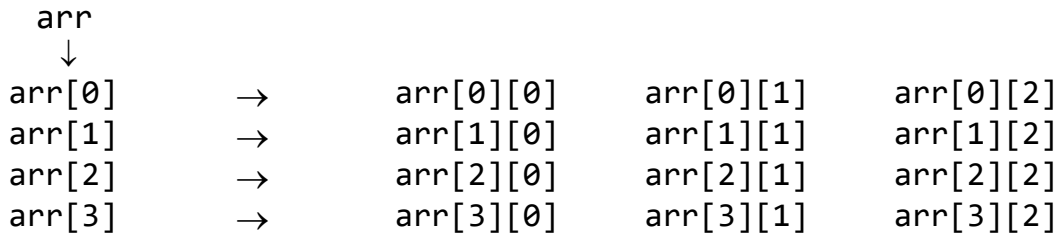
Результати роботи консольної програми:

```
Введіть 8 цілих чисел:
13   6   7   0   -3   8   -4   0
Кількість додатних елементів = 4
Сума парних елементів = 10
```

6.4 Вказівники на багатовимірні масиви

Вказівники на багатовимірні масиви у мові C++ – це масиви масивів, тобто такі масиви, елементами яких є масиви. При оголошенні таких масивів у пам'яті комп'ютера створюється декілька різних об'єктів. Приміром, при виконанні оголошення двовимірного масиву `int arr[4][3]` у пам'яті виділяється

ділянка для зберігання значення змінної `arr`, яка є вказівником на масив з чотирьох вказівників. Для цього масиву з чотирьох вказівників теж виділяється пам'ять. Кожний з цих чотирьох вказівників містить адресу масиву з трьох елементів типу `int`, і, отже, у пам'яті комп'ютера виділяється чотири ділянки для зберігання чотирьох масивів чисел типу `int`, кожна з яких складається з трьох елементів. Такий розподіл пам'яті показано на схемі:



Отже, оголошення `arr[4][3]` породжує в програмі три різних об'єкти: вказівник з ідентифікатором `arr`, безіменний масив з чотирьох вказівників і безіменний масив з дванадцяти чисел типу `int`. Для доступу до безіменних масивів використовуються адресні вирази з вказівником `arr`. Доступ до елементів масиву вказівників здійснюється із зазначенням одного індексного виразу у формі `arr[2]` чи `*(arr+2)`. Для доступу до елементів двовимірного масиву чисел типу `int` має бути використано два індексні вирази у формі `arr[1][2]` або еквівалентних їй `*(*(arr+1)+2)` та `*(arr+1)[2]`.

Нагадаємо, що елементи двовимірних масивів розміщуються у пам'яті підряд і між його рядками немає ніяких проміжків. Такий порядок дає можливість звертатися до будь-якого елемента багатовимірного масиву, використовуючи адресу його початкового елемента та лише один індексний вираз. Так, для матриці `arr` звертання `*(arr)` є посиланням на елемент `arr[0][0]`, звертання `*(arr+2)` – на елемент `arr[0][2]`, а звертання `*(arr+3)` – на елемент `arr[1][0]` і т. ін. Тобто, застосовуючи оператори циклу, можна організувати поелементне опрацювання елементів матриці, наприклад, введення усієї матриці з елемента `dataGridView1`:

```

for (int i=0; i<4; i++)
for (int j=0; j<3; j++)
    *(arr+i*4+j) = Convert.ToInt32(dataGridView1[j,i]->Value);

```

Тут вираз `*(arr+i*4+j)` є аналогічний до `arr[i][j]`.

Розміщення тривимірного масиву відбувається аналогічно й оголошення

```
double w[3][4][5];
```

породжує у програмі крім самого тривимірного масиву із шістдесяти елементів типу `double` масив з чотирьох вказівників на тип `double`, масив з трьох вказівників на масив вказівників на тип `double`, і вказівник на масив масивів вказівників на тип `double`.

Для звертання до елемента `w[2][3][4]` тривимірного масиву також можна використовувати вказівник, оголошений як `double *p=w[0][0]` з одним індексним виразом у формі `p[2*4*5+3*5+4]` чи `p[59]`.

Приклад 6.2 Ввести матрицю цілих чисел розміром 4×4 і транспонувати її.

Розв'язок. Щоби створити функцію, яка прийматиме у якості аргументу двовимірний масив, слід враховувати, що ім'я масиву опрацьовується як його адреса. Тому відповідний формальний параметр є вказівником, як і для одновимірних масивів. Секрет успіху криється у правильному оголошенні вказівника. Наведемо два різні способи такої організації функції до якої для транспонування передається матриця.

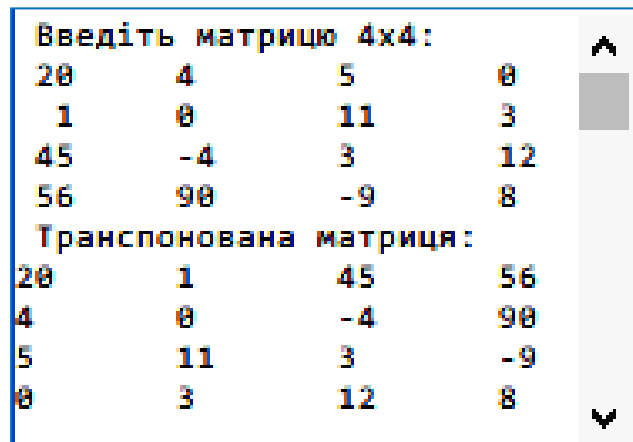
Перший спосіб розв'язання

У цьому способі, щоби кожного разу не зазначати розмірність квадратної матриці, створимо на початку відповідну константу n , і тоді зникає потреба передавання окремим параметром розмірності цієї матриці до функції.

У наведеній тут функції транспонування матриці `transp()` матриця передається до функції як вказівник і тоді більш звичне звертання до елементів матриці `u[i][j]` слід замінити виразом `*(u+n*i+j)` чи `u[n*i+j]`.

Текст функції та її виклику в основній програмі:

```
#include <iostream>
#include <conio.h>
using namespace std;
const int n=4;
// Функція транспонування матриці
void transp(int *u)
{ for (int i=0; i<n-1; i++)
  for (int j=i+1; j<n; j++)
  { int p= *(u+n*i+j);
    *(u+n*i+j) = *(u+n*j+i);
    *(u+n*j+i) = p;
  }
}
int main()
{
  setlocale(0, ".1251");
  system("color F0");
  int i,j, a[n][n];
  cout << " Введіть матрицю 4x4:\n";
  for (i=0; i<n; i++)
  for (j=0; j<n; j++) cin >> a[i][j];
  transp(&a[0][0]);
  cout << " Транспонована матриця:"<< endl;
  for (i=0; i<n; i++)
  { for (j=0; j<n; j++) cout<<a[i][j]<<"\t";
    cout << endl;
  }
  _getch();
  return 0;
}
```



Введіть матрицю 4x4:

20	4	5	0
1	0	11	3
45	-4	3	12
56	90	-9	8

Транспонована матриця:

20	1	45	56
4	0	-4	90
5	11	3	-9
0	3	12	8

Другий спосіб розв'язання

У цьому способі, на відміну від першого, передаватимемо розмірність квадратної матриці окремим параметром до функції. Крім того, використаємо ту особливість, що наша матриця `u` по суті є вказівником на масив з чотирьох значень типу `int`, і тому для функції є припустимим такий прототип:

```
void transp(int (*u)[4], int n);
```

У цьому запису дужки є обов'язковими, оскільки оголошення `int *u[4]` є масивом чотирьох вказівників на данні типу `int`, а параметр функції не може бути масивом. Існує альтернативний формат запису цього прототипу, який має таке саме тлумачення, але сприймається для розуміння краще:

```
void transp(int u[][4], int n);
```

Обидва варіанти означають, що параметр `u` є вказівником на масив чотирьох значень типу `int`, а не безпосередньо масивом. Число 4 у квадратних дужках задає кількість стовпців, а кількість рядків можна передавати окремим параметром.

Текст функції та її виклику в основній програмі:

```
#include <iostream>
#include <conio.h>
using namespace std;
const int n=4;
// Функція транспонування матриці
void transp(int (*u)[4], int n)
{ for (int i=0;i<n-1;i++)
  for (int j=i+1;j<n;j++)
  { int p=u[i][j];
    u[i][j]=u[j][i];
    u[j][i]=p;
  }
}
int main()
{
  setlocale(0, ".1251");
  system("color F0");
  int i,j, a[n][n];
  cout << " Введіть матрицю 4x4:\n";
  for (i=0; i<n; i++)
  for (j=0; j<n; j++) cin >> a[i][j];
  transp(a,n);
  cout << " Транспонована матриця:"<< endl;
  for (i=0; i<n; i++)
  { for (j=0; j<n; j++) cout<<a[i][j]<<"\t";
    cout << endl;
  }
  _getch();
  return 0;
}
```

Введіть матрицю 4x4:			
1	2	3	4
4	5	6	7
8	7	9	11
12	13	14	0
Транспонована матриця:			
1	4	8	12
2	5	7	13
3	6	9	14
4	7	11	0

Третій різновид передавання до функції двовимірних масивів, зазначений у п. 5.3.4, наведено у підрозд. 6.7 після того як буде розглянуто оператори динамічного виділення пам'яті для таких масивів.

6.5 Динамічна пам'ять

Окрім звичайної пам'яті (стека), в якій автоматично розміщуються змінні за їхнього оголошення, існує ще й *динамічна пам'ять* (heap – купа), в якій змінні можуть розміщуватися динамічно. Це означає, що пам'ять виділяється під час виконання програми, й лише тоді, коли у програмі зустрінеться спеціальна інструкція. Основна потреба у динамічному виділенні пам'яті виникає, коли розмір або кількість даних заздалегідь є невідомі, а визначаються в процесі виконання програми.

У C++ існує кілька команд для виділення динамічної пам'яті.

1) Найчастіше використовується оператор **new**, який у загальному вигляді записується як:

```
<тип> *<ім'я_вказівника> = new <тип>;
```

Приклади:

```
int *p = new int;
```

Тут виділяється місце в пам'яті під ціле число й адреса цієї ділянки пам'яті записується у змінну-вказівник p. Звернутися до цього числа можна буде через вказівник на нього: `*p = 2;`

```
int *p = new int(5);
```

Ця команда не лише виділить місце у пам'яті під ціле число, а й запише в цю ділянку пам'яті значення 5. Адреса першої комірки виділеної ділянки пам'яті присвоюється змінній-вказівнику p. Звернутися до числа, на яке вказує p, можна аналогічно до попереднього прикладу. Приміром, щоб збільшити таке число на 2, слід написати: `(*p) += 2;`

```
int *p = new int [5];
```

У цьому разі виділяється пам'ять під 5 цілих чисел, тобто фактично створюється так званий *динамічний масив* з 5-ти елементів. Звернутися до кожного з цих чисел можна за його номером: `p[0]`, `p[1]` і т. ін. або через вказівник: `*p` – те ж саме, що `p[0]`; `*(p+1)` – те ж саме, що `p[1]` тощо.

Пам'ять, яку було виділено динамічно, автоматично не звільнюється, тому програміст повинен обов'язково звільнити її самостійно за допомогою спеціальної команди. При виділенні пам'яті за допомогою оператора **new**, для звільнення пам'яті використовується **delete**:

```
delete <вказівник>;
```

Наприклад:

```
delete a;
```

Якщо оператором **new** було виділено пам'ять під кілька значень водночас (як у розглянутому вище прикладі 3), використовується форма команди **delete**

```
delete []<вказівник>;
```


Квадратні дужки повинні бути порожніми, операційна система контролює кількість виділеної пам'яті і при звільненні їй відома потрібна кількість байтів. Наприклад:

```
delete [] p;
```

Окрім операторів `new` та `delete`, існують функції, які перейшли до C++ із C, але вони використовуються на практиці набагато рідше.

2) Виділення динамічної пам'яті функцією `malloc` (від англ. «memory allocation» – виділення пам'яті), яка має такий формат:

```
void *malloc(size_t size);
```

Єдиний аргумент цієї функції `size` – кількість байтів, яку треба виділити. Функція повертає вказівник на початок виділеної пам'яті. Якщо для розміщення заданої кількості байтів є недостатньо пам'яті, функція `malloc()` повертає `NULL`. Вміст ділянки лишається незмінним, тобто там може залишитися «бруд». Якщо аргумент `size` дорівнює 0, функція повертає `NULL`. Наприклад, команда

```
int *a = (int*) malloc(sizeof(int));
```

виділяє пам'ять під ціле число й адресу початку цієї ділянки пам'яті записує у вказівник `a`.

Виділення пам'яті під 10 дійсних чисел за допомогою цієї функції:

```
double *a = (double*) malloc(sizeof(double)*10);
```

3) Виділення динамічної пам'яті функцією `calloc`, яка має такий формат:

```
void * calloc(size_t num, size_t size);
```

виділяє блок пам'яті розміром `num*size` (під `num` елементів по `size` байтів кожен) і повертає вказівник на виділений блок. Кожен елемент виділеного блока ініціалізується нульовим значенням (на відміну від функції `malloc`). Функція `calloc()` повертає `NULL`, якщо не вистачає пам'яті для виділення нового блока, або якщо значення `num` чи `size` дорівнюють 0.

Виділення пам'яті під 10 дійсних чисел за допомогою функції `calloc()`:

```
double *a = (double*) calloc(10, sizeof(double));
```

Змінити (зменшити чи то збільшити) розмір виділеного раніш блока динамічної пам'яті можна функцією `realloc`, яка має такий формат:

```
void *realloc(*b1, size);
```

при цьому розмір виділеного раніш блока пам'яті з адресою `*b1` змінюється на новий обсяг, який становитиме `size` байтів. Якщо зміна відбулася успішно, функція `realloc()` повертає вказівник на виділену ділянку пам'яті, а інакше повертається `NULL`. Якщо `*b1` є `NULL`, функція `realloc()` виконує такі самі дії, що й `malloc()`. Якщо `size` є 0, виділений за адресою `*b1` блок пам'яті звільняється – і функція повертає `NULL`. Для ілюстрації роботи функції `realloc()` надамо приклад змінення розміру раніш виділеної пам'яті під одновимірний масив з 10-ти дійсних елементів до 20-ти елементів:

```
a = (double *) realloc(a, 20);
```

Пам'ять, виділену за допомогою функцій `malloc()` і `calloc()`, слід звільнити за допомогою функції `free()`, наприклад:

```
free(a);
```

Використання згаданих функцій разом з вказівниками надає можливість керувати розподілом динамічної пам'яті.

Якщо не звільняти динамічно виділену пам'ять, коли вона стає більш не потрібною, у системі може виникнути нестача вільної пам'яті. Іноді це називають «витоком пам'яті».

6.6 Динамічні одновимірні масиви

Відмінності *динамічного масиву* від звичайного полягають у тому, що:

- ✓ пам'ять під динамічний масив виділяється динамічно за допомогою вищерозглянутих функцій;
- ✓ кількість елементів динамічного масиву може бути задано змінною (але у програмі її неодмінно має бути визначено до виділення пам'яті під масив).

Синтаксис оголошення динамічного одновимірного масиву за допомогою оператора `new` є такий:

```
<базовий тип> *<ім'я>=new <базовий тип>[<кількість елементів>];
```

Приклад оголошення дійсного динамічного масиву зі змінною кількістю елементів:

```
int N = richTextBox1->Lines->Length;  
double *a = new double [N];
```

Тут кількість елементів масиву є кількістю заповнених даними рядків у `richTextBox1` перед оголошенням масиву. Звільнення пам'яті від цього масиву `a` матиме вигляд:

```
delete []a;
```

Синтаксис оголошення динамічного одновимірного масиву за допомогою функції `malloc()` є такий:

```
<тип> *<ім'я> = (<тип>*) malloc(sizeof(<тип>)*<кільк. ел.>);
```

Приклад оголошення дійсного динамічного масиву зі змінною кількістю елементів за допомогою функції `malloc()`:

```
int N = richTextBox1->Lines->Length;  
double *a = (double *)malloc(sizeof(double)*N);
```

Приклад оголошення дійсного динамічного масиву зі змінною кількістю елементів за допомогою функції `calloc()`:

```
int N = richTextBox1->Lines->Length;  
double *a = (double *)calloc(sizeof(double), N);
```

Звільнення пам'яті з-під цього масиву `a`:

```
free (a);
```

Приклади програм з динамічними масивами

Приклад 6.3. Ввести ціле число N та масив з N дійсних чисел і створити новий масив з додатних елементів першого масиву.

Розв'язок. Оскільки кількість елементів заздалегідь невідома, тому пам'ять під обидва масиви слід виділяти динамічно. При розв'язанні подібних завдань зазвичай дотримуються такої послідовності дій:

1) визначити кількість елементів динамічного масиву, що формуватимуться;

2) оголосити динамічний масив(и) і виділити під нього пам'ять, використовуючи чи то `new`, чи `malloc()`, чи `calloc()`;

3) заповнити масив значеннями і вивести створений масив;

Наведемо два варіанти розв'язання цього завдання з використанням оператора `new` та функції `malloc()`.

Перший спосіб розв'язання з використанням оператора `new`:

```
#include <iostream>
using namespace std;
int main()
{ setlocale(0, ".1251");
  system("color F0");
  int i,n;
  cout << " Введіть кількість елементів масиву:\n"; cin>>n;
  // Виділення пам'яті під масив a; у цей момент ще не можна виділити пам'ять
  // під масив b, оскільки кількість його майбутніх елементів ще є невідома
  double *a=new double[n];
  cout << "Введіть "<<n<<" елементів масиву: ";
  for (i=0; i<n; i++) cin >> a[i];
  int k=0; // Кількість додатних елементів
  for (i=0; i<n; i++)
    if (a[i]>0) k++;
  // Виділення пам'яті під масив b
  double *b = new double[k];
  int j=0; // Індокси елементів нового масиву
  cout << "Новий масив з додатних елементів:\n";
  for (i=0; i<n; i++)
    if (a[i]>0)
    { b[j]=a[i];
      cout << b[j]<<"\t";
      j++;
    }
  // Звільнення пам'яті від масивів
  delete []a;
  delete []b;
  system("pause>>void");
  return 0;
}
```

Результати роботи консольної програми:

Введіть кількість елементів масиву: 10

Введіть 10 елементів масиву:

-3 5 0 -4 -55 1 9 33 6 11

Новий масив з додатних елементів:

5 1 9 33 6 11

Другий спосіб розв'язання з використанням функції `calloc()`:

```
#include <iostream>
using namespace std;
int main()
{
    setlocale(0, ".1251");
    system("color F0");
    int a[12], i, j1=0, j2=0;
    cout << "Введіть 12 цілих елементів масиву:\n";
    for (i=0; i<12; i++) cin >> a[i];
    int n1=0, n2=0;           // Кількість парних і непарних елементів
    for (i=0; i<12; i++)
        if (a[i]%2!=0) n1++; // Обчислення кількості непарних елементів n1
        else n2++;          // та кількості парних елементів n2.
    // Виділення пам'яті під масиви
    int *b1 = (int *)calloc(sizeof(int), n1);
    int *b2 = (int *)calloc(sizeof(int), n2);
    for (i=0; i<12; i++) // Формування масивів b1 та b2
        if (a[i]%2!=0) { b1[j1]=a[i]; j1++; }
        else { b2[j2]=a[i]; j2++; }
    // Виведення масивів в окремих циклах,
    // оскільки масиви мають різну кількість елементів
    cout << "Масив з парних елементів:\n";
    for (i=0; i<n1; i++) cout << b1[i] << "\t";
    cout << "\nМасив з непарних елементів:\n";
    for (i=0; i<n2; i++) cout << b2[i] << "\t";
    // Звільнення пам'яті від b1 та b2
    free (b1);
    free (b2);
    system("pause>>void");
    return 0;
}
```

Існує можливість звертатися до елементів масиву без індексації за допомогою вказівників. У циклі за допомогою індексу поточний елемент масиву записується як `a[i]`. Згадаймо, що ім'я масиву `a` можна використовувати як вказівник на початок (нульовий елемент) масиву. Тоді, згідно з арифметикою вказівників, `a+i` — це вказівник на елемент, який міститься на `i` комірок далі від початку масиву, тобто вказівник на `a[i]`. Значення елемента `a[i]` можна записати за допомогою операції розадресації: `*(a+i)`. Якщо у програмі замінити всі звертання до елементів масивів за допомогою індексів на звертання до елементів за допомогою вказівників, то фрагмент програми матиме вигляд:

```
for (i=0; i<12; i++) // Формування масивів b1 та b2
    if (*(a+i)%2 != 0) { *(b1+j1)=*(a+i); j1++; }
    else { *(b2+j2)=*(a+i); j2++; }
```

Приклад 6.4. Ввести масив з 12-ти цілих чисел і створити з нього два нових масиви: перший масив з непарних елементів, а другий – з парних.

Розв'язок. Кількість елементів вихідного масиву є відома (12), тому цей масив у програмі буде оголошено в звичайний спосіб: `int a[10]`; Два нових масиви `b1` та `b2` доцільно створити динамічно, оскільки кількість елементів цих масивів `n1` та `n2` буде обчислено у програмі.

Текст консольної програми:

```
#include <iostream>
using namespace std;
int main()
{
    setlocale(0, ".1251");
    system("color F0");
    int a[12], i, j1=0, j2=0;
    cout << "Введіть 12 цілих елементів масиву:\n";
    for (i=0; i<12; i++) cin >> a[i];
    int n1=0, n2=0;           // Кількість парних і непарних елементів
    for (i=0; i<12; i++)
        if (a[i]%2!=0) n1++; // Обчислення кількості непарних елементів n1
        else n2++;          // та кількості парних елементів n2.
    // Виділення пам'яті під масиви
    int *b1 = new int [n1], *b2 = new int [n2];
    for (i=0; i<12; i++) // Формування масивів b1 та b2
        if (a[i]%2!=0) { b1[j1]=a[i]; j1++; }
        else { b2[j2]=a[i]; j2++; }
    // Виведення масивів в окремих циклах,
    // оскільки масиви мають різну кількість елементів
    cout << "Масив з парних елементів:\n";
    for (i=0; i<n1; i++) cout << b1[i] << "\t";
    cout << "\nМасив з непарних елементів:\n";
    for (i=0; i<n2; i++) cout << b2[i] << "\t";
    // Звільнення пам'яті від b1 та b2
    delete []b1;
    delete []b2;
    system("pause>>void");
    return 0;
}
```

Результати роботи консольної програми:

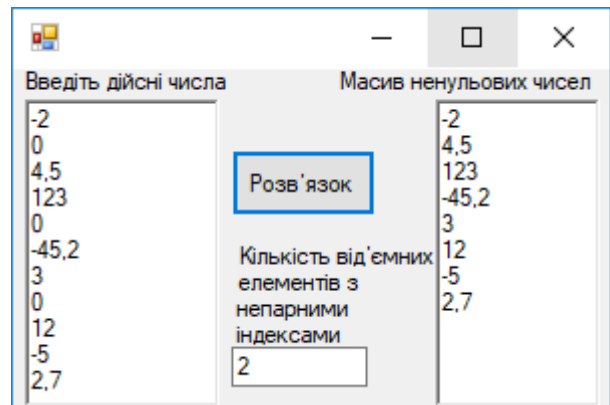
```
Введіть 12 цілих елементів масиву:
22 55 7 9 10 7 9 3 11 44 71 18
Масив з парних елементів:
22 10 44 18
Масив з непарних елементів:
55 7 9 7 9 3 11 71
```

Приклад 6.5. Ввести дійсні числа і створити з них масив ненульових чисел. Обчислити кількість від'ємних елементів цього масиву з непарними індексами.

Розв'язок. Оскільки заздалегідь невідомо кількість чисел, які будуть введені в елемент керування `richTextBox` на формі, тому має сенс організувати динамічний масив.

Текст програми з формою:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    int i, j=0;    // i - індекс числа richTextBox1, j - індекс елемента масиву,
    int k=0;       // змінна k для кількості від'ємних елементів масиву
    int n=richTextBox1->Lines->Length; // Кількість чисел в richTextBox1
    int N=0;       // Кількість ненульових чисел
    double x;
    for (i=0; i<n; i++) // Зчитування у циклі чисел з richTextBox1
    { x=Convert::ToDouble(richTextBox1->Lines[i]);
      if (x!=0) N++;    // й обчислення кількості ненульових чисел
    }
    double *a=new double[N]; // Виділення пам'яті під динамічний масив
    richTextBox2->Clear();
    for (i=0; i<n; i++)
    { x=Convert::ToDouble(richTextBox1->Lines[i]);
      if (x!=0)           // Записування ненульових чисел до масиву
      { a[j]=x;
        richTextBox2->Text += a[j].ToString() + "\n";
        j++;              // Збільшення індексу елемента масиву
      }
    }
    // Обчислення кількості від'ємних
    // елементів з непарними індексами
    for (i=0; i<N; i++)
        if (a[i]<0 && (i+1)%2!=0) k++;
    textBox1->Text=k.ToString();
    // Звільнення пам'яті від масиву
    delete []a;
}
```



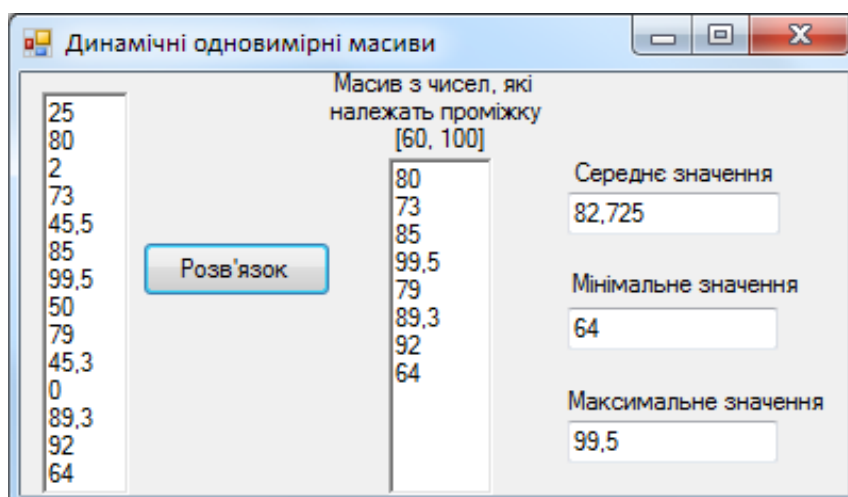
Приклад 6.6. Ввести деяку послідовність дійсних чисел і створити динамічний масив лише з чисел, значення яких попадає в проміжок $[60, 100]$. За допомогою функції визначити мінімальний і максимальний елементи та обчислити середнє арифметичне всіх елементів.

Розв'язок. Щодо практичного застосування уявімо, що початкова послідовність чисел є списком екзаменаційних оцінок, з якого слід відібрати лише ті, які задовольняють умові успішної задачі екзамену. А тому доречним є визначення середнього балу успішності та визначення найкращої і найгіршої екзаменаційної оцінки.

Програмний код:

```
double fun(double a[], int n, double &min, double &max)
{ double s=0; min=max=a[0];
  for (int i=0; i<n; i++)
  { s+=a[i];
    if (min>a[i]) min=a[i];
    if (max<a[i]) max=a[i];
  }
  return s/n;
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{ int kol=0, i, j;
  for (i=0; i<richTextBox1->Lines->Length; i++)
  { // Зчитування числа x з richTextBox1
    double x=Convert::ToDouble(richTextBox1->Lines[i]);
    if (x>=60 && x<=100) kol++; // kol - кількість елементів масиву
  }
  if (!kol) MessageBox::Show("Немає чисел з діапазону [60, 100]");
  else
  { double *a = new double[kol]; // Оголошення динамічного масиву
    // Вибірка елементів масиву з richTextBox1
    for (i=j=0; i<richTextBox1->Lines->Length; i++)
    { double x=Convert::ToDouble(richTextBox1->Lines[i]);
      if(x>=60 && x<=100) { a[j]=x; j++; }
    }
    richTextBox2->Clear();
    for (i=0; i<kol; i++) // Виведення елементів сформованого масиву
      richTextBox2->AppendText(a[i].ToString()+"\n");
    double sr, min, max;
    sr=fun(a,kol,min,max); // Виклик функції
    textBox1->Text=sr.ToString(); // Виведення результатів
    textBox2->Text=min.ToString();
    textBox3->Text=max.ToString();
    delete []a; // Звільнення динамічної пам'яті
  } }
```

Результати роботи:



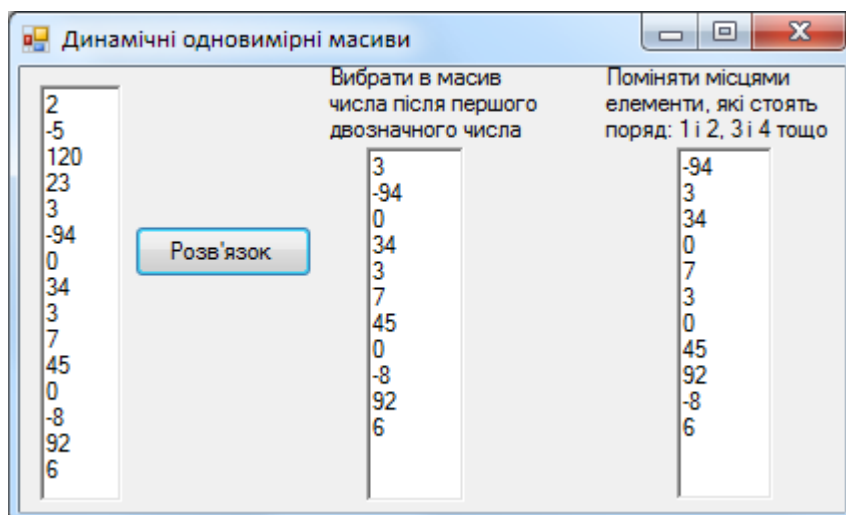
Приклад 6.7. Ввести деяку послідовність цілих чисел і створити динамічний масив з чисел, розміщених після першого двознакового числа (якщо двознакових чисел немає, вибрати всі). За допомогою функції поміняти місцями елементи, які стоять поряд: 1 і 2, 3 і 4 тощо.

Розв'язок. Якщо організувати створення програмного проекту з формою, це дозволить одночасно бачити введену послідовність чисел (в richTextBox1), сформований масив (в richTextBox2) і масив з переставленими елементами (в richTextBox3).

Програмний код:

```
void f(int a[], int n)
{ if (n%2) n--; // Якщо кількість елементів непарна, не розглядати останній з них
  for (int i=0; i<n; i+=2)
  { int tmp=a[i];
    a[i]=a[i+1];
    a[i+1]=tmp;
  } }
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
  int n=0, i, kol;
  for (i=0; i<richTextBox1->Lines->Length; i++)
  { // Зчитування числа x з richTextBox1. Оскільки двознаковими числами можуть
    // бути і від'ємні числа, для спрощення наступної умови беремо модуль числа
    int x=Math::Abs(Convert::ToInt32(richTextBox1->Lines[i]));
    if (x<10 || x>99) continue;
    else { n=i+1; break;} // n - номер першого двознакового числа
  }
  if (!n) MessageBox::Show("Немає двознакових чисел!");
  else
  { // Обчислення kol - кількості елементів масиву
    kol = richTextBox1->Lines->Length - n;
    // Оголошення динамічного масиву
    int *a = new int[kol];
    // Введення елементів масиву з richTextBox1, починаючи з рядка n
    for (i=0; i<kol; i++)
      a[i]=Convert::ToInt32(richTextBox1->Lines[i+n]);
    // Виведення елементів сформованого динамічного масиву
    richTextBox2->Clear();
    for (i=0; i<kol; i++)
      richTextBox2->AppendText(a[i].ToString()+"\n");
    f(a,kol); // Виклик функції для переставлення елементів масиву
    // Виведення масиву з переставленими елементами
    richTextBox3->Clear();
    for (i=0; i<kol; i++)
      richTextBox3->AppendText(a[i].ToString()+"\n");
    // Звільнення динамічної пам'яті
    delete []a;
  }
}
```


Результати роботи:



Приклад 6.8. Ввести деяку послідовність цілих чисел і створити динамічний масив лише з чисел, розміщених до першого числа, яке має серед своїх цифр хоча б одну цифру 7. За допомогою функції обчислити середнє значення одноцифрових елементів масиву.

Програмний код:

```
double fun(int a[], int n)
{
    double s=0; int k=0;
    for(int i=0; i<n; i++)
        if(a[i]>=-9 && a[i]<=9) // перевірка: число одноцифрове?
            { s+=a[i]; k++; }
    if(k) s/=k;
    return s;
}

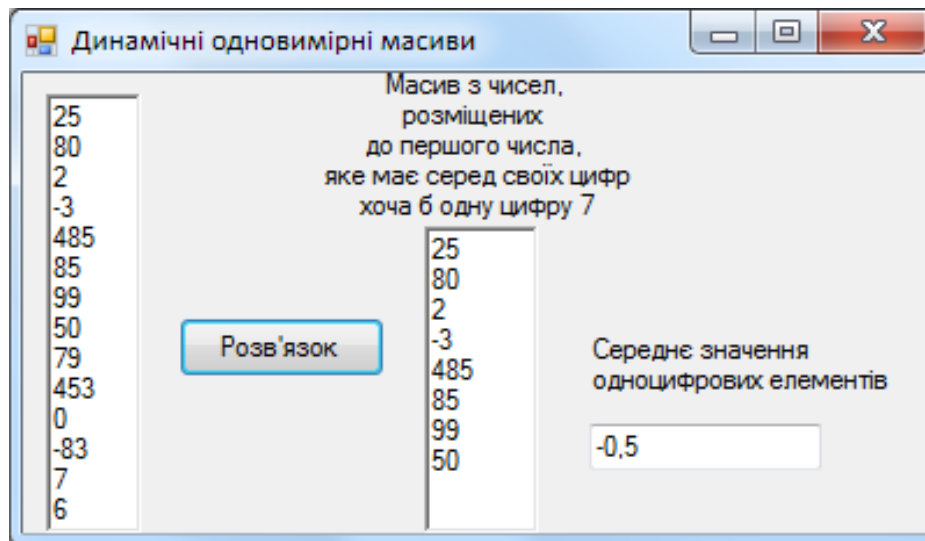
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    int kol=0, i, n; // kol - кількість елементів масиву
    for (i=0; i<richTextBox1->Lines->Length; i++)
    { // Зчитування числа x з richTextBox1
        int x=Convert::ToInt32(richTextBox1->Lines[i]);
        bool y=false;
        // Пошук сімок у числі. Якщо число має сімку, то kol - номер числа
        while (x)
        { n = x%10; // виділення окремої цифри числа в n
            if (n==7) { y=true; kol=i; break; }
            x /= 10;
        }
        if(y) break; // Якщо число має сімку, не перевіряти решту чисел
    }
    if (!kol) MessageBox::Show("Немає чисел із сімками");
    else
```

```

{
    int *a = new int[kol]; // Оголошення динамічного масиву
    // Введення елементів масиву з richTextBox1 до першого числа із сімкою
    for (i=0; i<kol; i++)
        a[i]=Convert::ToInt32(richTextBox1->Lines[i]);
    // Виведення елементів сформованого масиву
    richTextBox2->Clear();
    for (i=0; i<kol; i++)
        richTextBox2->AppendText(a[i].ToString()+"\n");
    double sr=fun(a,kol); // Виклик функції
    textBox1->Text=sr.ToString(); // Виведення результатів
    delete []a; // Звільнення динамічної пам'яті
}
}

```

Результати роботи:



Приклад 6.9. Надати можливість введення цілого числа (від 1 до 10) та динамічно створити і розмістити на формі відповідну кількість елементів керування для введення елементів динамічно створюваного масиву і подальшого обчислення суми додатних елементів цього масиву.

Розв'язок. Як приклад динамічного створювання та розміщування елементів керування на формі розглянемо створення певної кількості `textBox`-ів та `label`-ів. Причому наведемо два різні підходи для роботи зі створеними елементами керування як з масивом і як з окремими об'єктами. Аналогічним чином можна працювати з будь-якими елементами керування.

Окрім зазначених алгоритмів динамічного створювання елементів керування, у цій програмі застосовано діалогове вікно `InputDialog()` для введення цілого числа, яке визначено у бібліотеці `Microsoft.VisualBasic.dll`, і тому слід долучивши відповідні файли вгорі програмного коду:

```

#using <system.windows.forms.dll>
#using <Microsoft.VisualBasic.dll>

```

Синтаксис `InputBox()` такий:

```
static String^ InputBox(String^ Prompt, String^ Title,  
    String^ DefaultResponse, int XPos, int YPos);
```

де `Title` – текст надпису вікна;

`Prompt` – пояснювальний текст (підказка) у вікні;

`DefaultResponse` – пропонуване значення для введення, з яким можна погодитись або змінити його на нове значення;

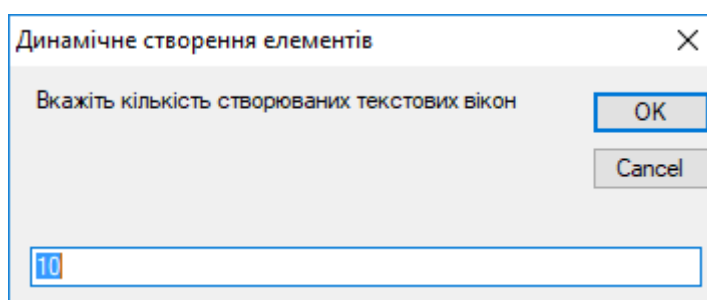
`XPos` – відступ від лівого краю екрана;

`YPos` – відступ від верхнього краю екрана.

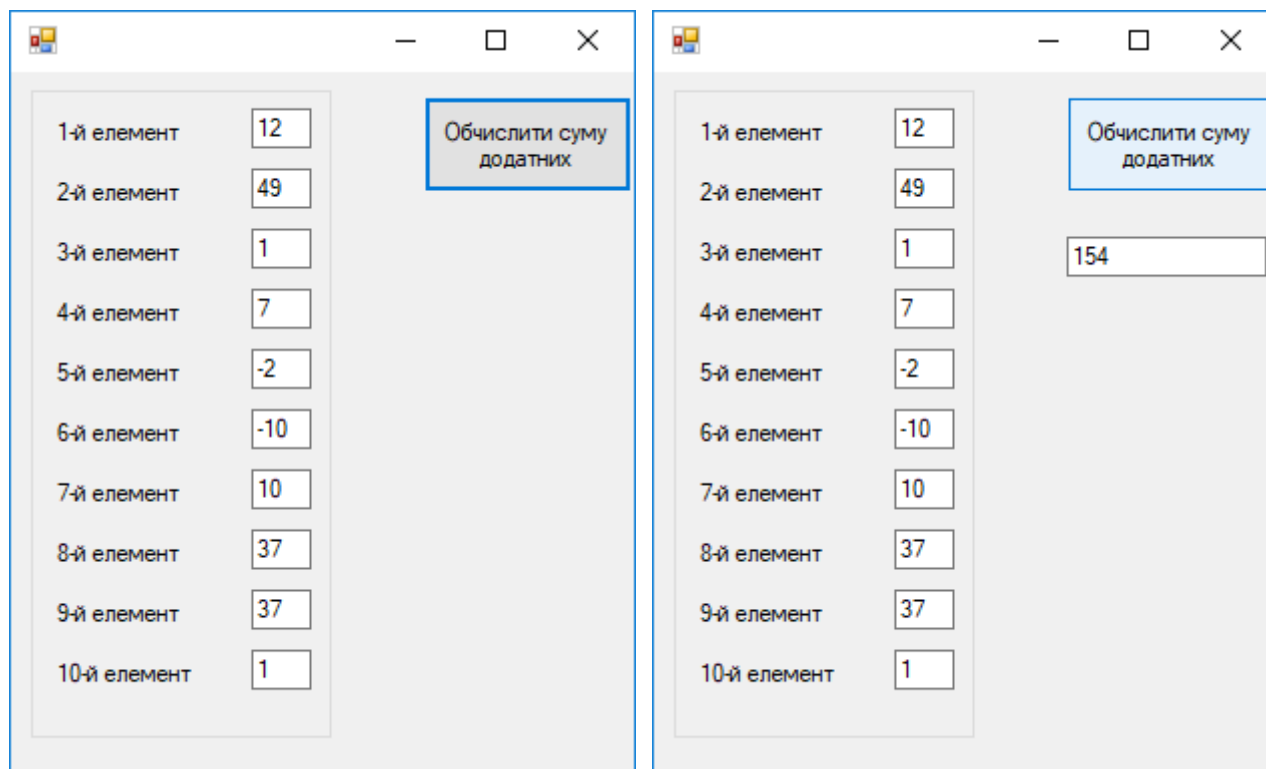
Вікно введення, створене у програмі командою

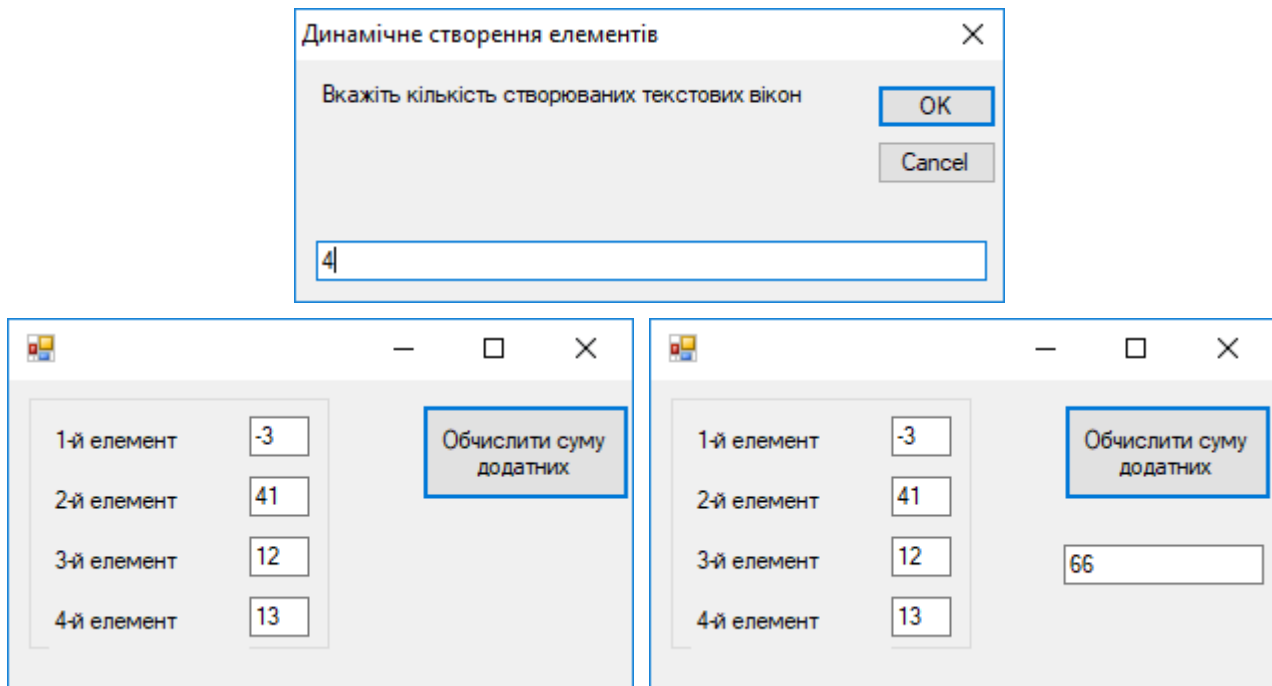
```
String^s= Microsoft::VisualBasic::Interaction::InputBox("Вкажіть кількість  
    створюваних текстових вікон", "Динамічне створення елементів", "10", 250, 250);
```

матиме вигляд:



Приклад вигляду форми з відповідною кількістю динамічно створених елементів керування на ній наведено ліворуч, а праворуч від нього показано вигляд цієї форми після натискання кнопки й обчислення суми додатних елементів масиву.





Текст програми

```
#pragma once
#using <system.windows.forms.dll>
#using <Microsoft.VisualBasic.dll>
. . . . .
int count; // Кількість елементів масиву
static array<TextBox>^ tb=gcnew array<TextBox>(10);
GroupBox^ GroupBox1; // Масив десяти вказівників на клас TextBox

private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
    ww:
    String^s= Microsoft::VisualBasic::Interaction::InputBox("Вкажіть кількість
    створюваних текстових вікон","Динамічне створення елементів","10",250,250);
    if (s!="") count=Convert::ToInt32(s); else return;
    if (count>10 || count<=0)
    { MessageBox::Show("Введено некоректне значення!");
      goto ww;
    }
    // Створити новий елемент керування класу GroupBox на формі
    GroupBox1 = gcnew GroupBox();
    GroupBox1->Location = Point(10,3); // Розміщення щодо лівого краю форми
    this->Controls->Add(GroupBox1);
    // Встановлення висоти відповідно до кількості створених елементів
    GroupBox1->Size = System::Drawing::Size(150,count*33);
    Form1::Height=GroupBox1->Height+60; // Висота форми
    Random^ rnd = gcnew Random();
    for (int i=0; i<count; i++)
    {
        Label^ L = gcnew Label(); // Створити новий елемент класу Label на формі
```

```

L->Location = Point(10,i*30+20);// Розміщення щодо верхнього краю форми
this->Controls->Add(L);
// Для розміщення створених елементів задається їхній предок - елемент GroupBox1
L->Parent=GroupBox1;
L->Text=(i+1).ToString()+"-й елемент"; // Надпис мітки
tb[i] = gcnew TextBox();
// Створити новий елемент керування класу TextBox на формі як елемент масиву td
tb[i]->Size = System::Drawing::Size(30,60); // Розміри елемента
tb[i]->Location = Point(110,i*30+15);
this->Controls->Add(tb[i]);
tb[i]->Parent=GroupBox1;
tb[i]->Text=rnd->Next(-10,50).ToString(); // Випадкове число
Controls->Remove(L); // Очищення пам'яті від L
}
textBox1->Visible=false;
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    double S=0, *a = new double [count];
    for (int i=0; i<count; i++)
    { a[i]=Convert::ToDouble(tb[i]->Text);
      if (a[i]>0) S+=a[i];
    }
    textBox1->Visible=true;
    textBox1->Text=S.ToString();
    delete []a;
}
private: System::Void Form1_FormClosing(System::Object^ sender,
System::Windows::Forms::FormClosingEventArgs^ e)
{
    for (int i=0; i<count; i++) Controls->Remove(tb[i]);
    Controls->Remove(GroupBox1);
}

```

6.7 Динамічні двовимірні масиви (матриці)

Основний спосіб роботи з динамічними матрицями базується на використанні подвійних вказівників. Для того, щоб працювати з динамічними матрицями, доцільно представити матрицю як масив векторів (масив, що містить вказівники на рядки матриці). Перед початком роботи з такою матрицею треба спочатку виділити пам'ять під масив вказівників на рядки, а лише тоді виділяти пам'ять для зберігання самих даних. Наприклад, оголошення динамічної матриці 3×58 дійсних чисел може бути таким:

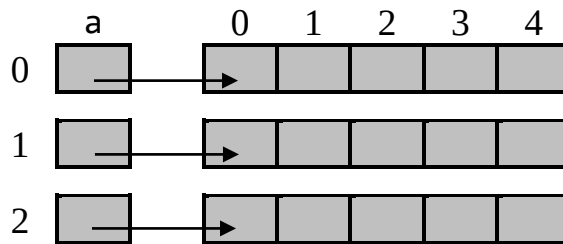
```

double **a = new double*[3]; // 3 рядки матриці
for (int i=0; i<3; i++)
    a[i] = new double[5]; // по 5 стовпців

```

Аналогічне оголошення за допомогою `calloc()`:

```
double** a = (double**) calloc (3, sizeof(double*));
for (int i=0; i<3; i++)
    a[i] = (double *) calloc (5, sizeof(double));
```



При цьому буде виділено пам'ять під кожний рядок матриці окремо, тобто буде утворено три різні одновимірні масиви. Адреси нульових елементів цих масивів зберігатимуться в допоміжному масиві `a` (пам'ять під нього слід виділити заздалегідь). Елементами цього масиву будуть адреси дійсних чисел, тому вони матимуть тип «вказівник на дійсне число», тобто `double*`.

Після використання матриці виділена за допомогою `new` для неї пам'ять звільняється у зворотній послідовності:

```
for (int i=0; i<3; i++) delete [] a[i];
delete [] a;
```

Тобто при виділенні пам'яті для цієї матриці спочатку оголошується вказівник другого порядку `double **a`, який посилається на масив вказівників `double*[7]` (де 7 – розмір масиву). Після цього в циклі для кожного рядка матриці виділяється по вісім елементів.

Звертання до елементів динамічної матриці здійснюється так само, як і до елементів статичної матриці – `a[i][j]`.

Приклад 6.10. Ввести кількість рядків і стовпців матриці та заповнити елементи цієї матриці випадковими числами з діапазону від 1 до 100. Вивести на екран елементи самої матриці та обчислити добуток її додатних елементів.

Розв'язок. Оскільки кількість рядків та стовпчиків матриці заздалегідь є невідомі, матрицю доцільно організувати динамічно. Пам'ять під кожний рядок матриці виділятимемо окремо. При роботі з елементами матриці явно зазначатимемо обидва індекси (як при роботі з елементами нединамічної матриці), що вважається більш зручним.

Для генерування випадкових чисел використовуватимемо функцію `rand()` (див. підрозд. 3.6). Перед використанням цієї функції треба ініціалізувати генератор випадкових чисел функцією `srand()` для того, щоб кожного разу формувалися різні, а не одні й ті самі числа. Аргументом функції `srand()` можна задати функцію `time()`, викликану з нульовим вказівником типу `time_t`, яка повертає кількість секунд від 1 січня 1970 року і тим самим задає випадкове зерно для генерації чисел. Функції `rand()` та `srand()` прописані в заголовному файлі `<stdlib>`, а `time()` – у файлі `<ctime>`. Наприклад, команди

```
time_t t;
srand((unsigned) time(&t));
for (int i=0; i<10; i++) printf("%d\n", rand() % 100);
```

генеруватимуть послідовність з 10-ти випадкових цілих чисел від 0 до 99. Подібна конструкція для дійсних чисел від 0 до 10 може бути такою:

```
srand(time(NULL));           // можна і так: srand(time(0));
for (int i=0; i<10; i++)
    cout<<setw(4)<<setprecision(2)<<(rand()%101)/double((rand()%10))<<endl;
```

Приклад програми з динамічним виділенням пам'яті для матриці випадкових чисел, розмірність якої ($m \times n$) вводить користувач:

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <iomanip>
using namespace std;
int main()
{ setlocale(0, ".1251");
  srand(time(NULL));           // генерація випадкових чисел
  int m, n;                    // змінні для розмірності матриці
  cout<<"Введіть розмірність матриці через пробіл: "; cin>>m>>n;
  // динамічне створення матриці дійсних чисел розміром m*n
  double **a = new double*[m], p=1; // n рядків матриці
  for (int i=0; i<m; i++) a[i] = new double[n]; // по m стовпців
  // заповнення матриці випадковими дійсними числами від 1 до 10
  for (int i=0; i<m; i++)
    for (int j=0; j<n; j++)
      a[i][j] = (rand() % 101 + 1) / double((rand() % 10 + 1));
  // виведення матриці
  cout<<"Матриця: "; ;
  for (int i=0; i<m; i++)
  { for (int j=0; j<n; j++)
    cout << setw(4) << setprecision(2) << a[i][j] << " ";
    if (a[i][j]>0) p*=a[i][j]; // Обчислення добутку додатних елементів
    cout << endl;
  }
  cout<<"Добуток додатних елементів = "<< p << endl;
  for (int i=0; i<m; i++) delete []a[i]; // звільнення динамічної пам'яті
  delete []a;
  system("pause>>void");
  return 0;
}
```

```
Введіть розмірність матриці через пробіл:  3  7
Матриця:
-3.4  0    4.3  2.7  -7.1  0.3  -5    9    -1
2    -9.4  1.5  6    1.2  -6.8  2.1  1.1  5.8
1.2  4.7  3    -6.4  -0.4  -5    -3.9  3    2.1
Добуток додатних елементів =
```

При виведенні цієї матриці було використано модифікатор `setw`, який дозволив задавати ширину (кількість символівних позицій) кожного виведеного числа. Так, модифікатор `setw(4)` для кожного елемента відвів максимум по чо-

тири позиції, що дозволило вирівняти по стовпцях елементи різного розміру. Якщо елемент мав меншу, ніж чотири знаки ширину, перед ним було виведено пробіли. Використаний при виведенні маніпулятор `setprecision(2)` дозволив обмежити кількість знаків після десяткової крапки максимум двома знаками. Саме для цих засобів було долучено файл `iomanip`, а докладно вони були розглянуті у п. 5.3.2.

Приклад 6.11. Ввести цілочисельну матрицю 6×4 і створити нову матрицю з тих рядків введеної матриці, які не містять нульових елементів.

Розв'язок. На відміну від введеної матриці, створювана матриця буде динамічною, оскільки кількість її рядків заздалегідь є невідома. Для обчислення кількості нульових елементів кожного рядка організуємо тимчасовий допоміжний масив `r`. Кількість рядків створюваної матриці визначатимемо як кількість ненульових елементів масиву `r` й у нову матрицю будемо копіювати лише ті рядки введеної, для яких `r=0`.

Програмний код:

```
#include <iostream>
using namespace std;
int main()
{ system("chcp 1251 > null");
  system("color F0");
  int a[6][4], i, j, rowc=0, k=0;
  cout<<"Введіть матрицю з 6-ти рядків і 4-х стовпців:"<<endl;
  int r[6]={0}; // r - масив кількості нульових елементів рядків матриці
  for (i=0; i<6; i++)
    for (j=0; j<4; j++)
    { cin>>a[i][j];
      if (a[i][j]==0) r[i]++;
    }
  for (i=0; i<6; i++) // Обчислення кількості рядків без нульових елементів
    if (r[i]==0) rowc++; // r[i]=0 означає, що в i-му рядку немає нулів
  int **b=new int *[rowc]; // Виділення пам'яті для динамічної матриці b
  for (i=0; i<rowc; i++) b[i]=new int [4];
  for (i=0; i<6; i++)
    if (r[i]==0) // Якщо в i-му рядку нулів немає, відбувається
    { for (j=0; j<4; j++) b[k][j]=a[i][j]; // копіювання рядка у нову матрицю
      k++; // і збільшення індексу рядка нової матриці.
    }
  cout<<"\nМатриця, в рядках якої немає нульових елементів:"<<endl;
  for (i=0; i<rowc; i++)
  { for (j=0; j<4; j++) cout<<b[i][j]<<"\t";
    cout << endl;
  }
  // Звільнення пам'яті від динамічної матриці
  for (int i=0; i<rowc; i++) delete [] b[i];
  delete []b;
```

```

system ("pause>>void");
return 0;
}

```

Результати роботи:

```

Введіть матрицю з 6-ти рядків і 4-х стовпців:
23      7      90      0
-6      0      88      45
33      78      9      4
12      4      8      90
0       3      0      -7
50      7      9      7

Матриця, в рядках якої немає нульових елементів:
33      78      9      4
12      4      8      90
50      7      9      7

```

Приклад 6.12. Ввести матрицю 3×7 дійсних чисел та створити нову матрицю з тих стовпців введеної матриці, які містять нульові елементи.

Програмний код:

```

#include <iostream>
using namespace std;
int main()
{ system("chcp 1251 > null");
  system("color F0");
  double a[3][7]; int i, j;
  cout<<"Введіть матрицю з 3-х рядків і 7-ми стовпців:"<<endl;
  for(i=0; i<3; i++)
    for(j=0; j<7; j++) cin>>a[i][j];
  int col=0; // col - кількість стовпців з нульовими елементами
  int r[7]={0}; // r - масив кількості нульових елементів стовпців матриці
  for(j=0; j<7; j++)
  { for(i=0; i<3; i++) // Обчислення
    if(a[i][j]==0) r[j]++; // кількості нулів у стовпці та
    if(r[j]!=0) col++; // кількості стовпців з нульовими елементами
  }
  double **b=new double*[3]; // Виділення пам'яті
  for(i=0; i<3; i++) // для динамічної матриці
    b[i]=new double [col];
  int k=0; // k - індекс стовпця нової матриці (з нульовими елементами)
  for(j=0; j<7; j++)
    if(r[j]!=0) // Якщо в j-му стовпці є нулі, скопіювати
    { for(i=0; i<3; i++) b[i][k]=a[i][j]; // стовпець у нову матрицю
      k++; // і збільшити індекс стовпця нової матриці
    }
  if(!col) cout<<"\nНемає стовпців з нульовими елементами";
  else
  cout<<endl<<col<<" стовпці(в) з нульовими елементами";
}

```

```

cout<<"\nМатриця, у стовпцях якої є нульові елементи:"<<endl;
for(i=0; i<3; i++)
{ for(j=0; j<col; j++) cout<<b[i][j]<<"\t";
  cout << endl;
}
for(int i=0; i<3; i++)
    delete [] b[i];      // звільнення пам'яті від динамічної матриці
delete []b;
system ("pause>>void");
return 0;
}

```

Результати роботи:

```

Введіть матрицю з 3-х рядків і 7-ми стовпців:
45      7.8      0      -7      2.8      30      -5
0       5.8      5.7      0       12      -7      0
0.1     -5       23      6        0       4.8     41

5 стовпці(в) з нульовими елементами
Матриця, у стовпцях якої є нульові елементи:
45      0      -7      2.8      -5
0       5.7      0       12      0
0.1     23      6        0       41

```

Приклад 6.13. Обчислити визначник (детермінант) квадратної матриці розмірністю $n \times n$, значення n ввести.

Розв'язок. Обчислення визначника матриці є поширеною задачею вищої математики. Так, без значення визначника матриці не обійтися при розв'язанні складних систем рівнянь. За допомогою детермінанта визначають наявність та єдиність існування розв'язку систем рівнянь. Важко переоцінити важливість уміння правильно і точно знаходити визначник матриці в математиці. Методи обчислення визначників є теоретично простими, проте зі збільшенням розміру матриці обчислення стають дуже громіздкими і вимагають величезної уваги і багато часу.

Для матриці першого порядку значення визначника дорівнює єдиному елементу цієї матриці: $\Delta = |a_{11}| = a_{11}$.

Для матриці другого порядку (2×2) значення визначника обчислюється як:

$$\Delta = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11} \cdot a_{22} - a_{12} \cdot a_{21}.$$

Для матриць більш високих порядків (вище другого) $n \times n$ визначник можна обчислити, застосувавши *рекурсивну формулу*, яку ще називають *розкладанням по рядку*:

$$\Delta = \begin{vmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{n1} & \dots & a_{nn} \end{vmatrix} = \sum_{j=1}^n (-1)^{1+j} \cdot a_{1j} \cdot M_j^1,$$

де M_j^1 – додатковий мінор до елемента a_{1j} – визначник матриці, отриманий з вихідної викресленням 1-го рядка та j -го стовпця.

Наприклад, формула обчислення визначника матриці 3-го порядку має вигляд:

$$\Delta = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11} \cdot \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{21} \cdot \begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} + a_{31} \cdot \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix}$$

Тобто, визначник порядку n розкладається на добутки визначників порядку $(n - 1)$ і коефіцієнтів, якими є елементи рядка (стовпця), за яким розкладалися мінори. Обчислені визначники $(n - 1)$ -го порядку обчислюються рекурсивно.

Один з найшвидших методів полягає у простій модифікації методу Гаусса. Дотримуючись *методу Гаусса*, довільну матрицю можна привести до ступінчастого вигляду (верхньотрикутної матриці), використовуючи лише дві операції над матрицею – перестановку двох рядків і додавання до одного з рядків матриці іншого рядка, помноженого на довільне число. З властивостей визначника випливає, що друга операція не змінює визначника матриці, а перша лише змінює його знак на протилежний. Визначник матриці, приведенної до ступінчастого вигляду, дорівнює добутку елементів її діагоналі, оскільки вона є трикутною. Метод Гаусса обчислення визначника добре працює для матриць порядку вище 3 на 3.

Далі у програмному коді наведемо обидва методи обчислення визначника у вигляді окремих функцій: ітеративний метод Гаусса і рекурсивний метод розкладання по рядку.

Визначник матриці NxN

Задайте розмір квадратної матриці: 3

Прийняти розмір матриці

Введіть матрицю або скористайтесь кнопкою для заповнення випадковими числами

Заповнити випадковими числами

Обчислити визначник

Метод Гаусса: 75

Рекурсивний: 75

9	9	3
9	-5	-2
-10	1	0

Визначник матриці NxN

Задайте розмір квадратної матриці: 7

Прийняти розмір матриці

Введіть матрицю або скористайтесь кнопкою для заповнення випадковими числами

Заповнити випадковими числами

Обчислити визначник

Метод Гаусса: 22481,6

Рекурсивний: 22481,6

0	6,6	1	6	-8	0	1,5
8	2	-7	0,5	6	10	4
-7	4	1	7	0	-3	0
1	3	0	1	0	-0,2	5
-5	-0,25	0	-2	1	7	-3
0	2	-7	0	1	-2	0
-3	1	-2	1,5	9	-1	5

Програмний код:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    int n = Convert::ToInt32(textBox1->Text); // Розмір квадратної матриці
    if (n <= 0)
    {
        MessageBox::Show("Некоректне значення! Введіть новий розмір квадратної матриці");
        return;
    }
    else
    {
        dataGridView1->Rows->Clear(); // Очистити (видалити) рядки.
        // Зробити видимими елементи на формі (при створенні форми у цих
        button2->Visible = true; button3->Visible = true; // елементів було
        dataGridView1->Visible = true; // "вимкнено" властивість видимості).
        label2->Visible = true; label3->Visible = true; label4->Visible = true;
        textBox2->Visible = true; textBox3->Visible = true;
        dataGridView1->ColumnCount = n; // Кількість стовпців.
        // "Вимкнути" режим автоматичного додавання рядків.
        dataGridView1->AllowUserToAddRows = false;
        dataGridView1->ColumnHeadersVisible = false; // Прибрати фіксовані
        dataGridView1->RowHeadersVisible = false; // стовпець і рядок.
        // Задати ширину кожного стовпця 50 пікселів.
        for (int i = 0; i < n; i++) dataGridView1->Columns[i]->Width = 50;
        dataGridView1->Rows->Add(n); // Вставити n рядків.
        dataGridView1->Height = n * 21 + 10; // Задати потрібні висоту і
        dataGridView1->Width = n * 50 + 10; // ширину елемента dataGridView.
    }
}

private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e)
{
    int n = Convert::ToInt32(textBox1->Text); // Розмір квадратної матриці.
    Random^ z = gcnew Random();
    for (int i=0; i<n; i++)
    for (int j=0; j<n; j++)
        dataGridView1->Rows[i]->Cells[j]->Value=Convert::ToString(z->Next(-10,11));
}

double detGauss(double **a, int n)
{
    int i,j,t,k,p=0; double d;
    for (i=0; i<n-1; i++)
    {
        t=1;
        while (a[i][i]==0)
        {
            for (j=0; j<n; j++)
            {
                d = a[i][j];
                a[i][j] = a[i+t][j];
                a[i+t][j] = d;
            }
        }
    }
}
```

```

    p++;
    t++;
}
for (k=i+1; k<n; k++)
{
    d = a[k][i]/a[i][i];
    for (j=0; j<n; j++)
        a[k][j] -= a[i][j]*d;
}
}
d = Math::Pow(-1,p);
for (i=0; i<n; i++) d*=a[i][i];
return d;
}
double detR(double** a, int n)
{
    int i,j; double d=0, **matr;
    if (n==1) d=a[0][0];
    else
        if (n==2) d=a[0][0]*a[1][1]-a[0][1]*a[1][0];
        else
        {
            matr = new double*[n-1];
            for (i=0; i<n; ++i)
            {
                for (j=0; j<n-1; ++j)
                {
                    if (j<i) matr[j]=a[j];
                    else     matr[j]=a[j+1];
                }
                d += Math::Pow(-1, (i+j)) * detR(matr,n-1) * a[i][n-1];
            }
            delete[] matr;
        }
    return d;
}
private: System::Void button4_Click(System::Object^ sender, System::EventArgs^ e)
{
    int i,j,n = Convert::ToInt32(textBox1->Text); // Розмір квадратної матриці
    double **a=new double *[n]; // Виділення пам'яті для динамічної матриці a
    for (i=0; i<n; i++) a[i]=new double [n];

    for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        a[i][j] = Convert::ToDouble( dataGridView1->Rows[i]->Cells[j]->Value);
    textBox2->Text = (detR(a, n)).ToString();
    textBox3->Text = (detGauss(a,n)).ToString();
}

```

Питання та завдання для самоконтролю

- 1) Що являє собою пам'ять комп'ютера?
- 2) Що таке адреса комірки?
- 3) Що таке вказівник?
- 4) Навести синтаксис оголошення вказівника?
- 5) Яке тлумачення має значення вказівника NULL?
- 6) В який спосіб можна отримати адресу змінної?
- 7) Яке призначення операції &?
- 8) В який спосіб можна дізнатися значення, на яке посилається вказівник?
- 9) Пояснити відмінності поміж змінними `a` та `b`, оголошення яких має вигляд:
 - а) `int a; int b;`
 - б) `int *a; double *b;`
- 10) Як можна виділити пам'ять під цілу змінну?
- 11) В який спосіб можна звільнити пам'ять, виділену за допомогою оператора `new`?
- 12) Як можна ініціалізувати пам'ять, виділену за допомогою оператора `new`?
- 13) В який спосіб можна звернутися до даних, для яких пам'ять було виділено динамічно?
- 14) Пояснити відмінності динамічного масиву від звичайного.
- 15) Вибрати правильні оголошення динамічного масиву з 5-ти цілих чисел:
 - а) `int a[5];`
 - б) `int *a[5];`
 - в) `int *a=new int [5];`
 - г) `int *a=new [5];`
 - д) `int *a=malloc(5);`
 - е) `int *a=(int*) malloc(20);`
 - ж) `int *a=new int (5);`
 - з) `int *a=(int*) malloc(5*sizeof(int));`
- 16) Припустимо, у програмі оголошено масив: `int *a=new int[7];`
Вибрати правильні команди і пояснити, що вони виконують:
 - а) `a=5;`
 - б) `a++;`
 - в) `a[3]++;`
 - г) `*a=1;`
 - д) `*(a+4)=3;`
 - е) `*a[3]=2;`
- 17) Вибрати правильні команди звільнення пам'яті від динамічного масиву `a` з 5-ти елементів: 14)
 - а) `delete a[5];`
 - б) `delete a[];`
 - в) `delete []a;`
 - г) `free (a);`
 - д) `free a[5];`
- 18) Припустимо, у програмі є такі оголошення:
`int *a; double *b; int *c(10); int d;`
Які з наведених операцій присвоювання є помилковими і чому:
 - а) `a=5;`
 - б) `b=a;`
 - в) `c=a;`
 - г) `*b=*c;`
 - д) `*a=3;`
 - е) `*c=a;`
 - ж) `a=d;`
 - з) `a=&d;`
 - и) `b=&d;`

19) Розглянемо масив a:

-5	30	8	-1	4

Яке значення має кожен з наступних виразів:

a) *a

b) *(a+4)

20) Припустімо, у програмі оголошено масив: `int *a=new int[4];`

Вибрати команди, які призведуть до помилки, і поясніть:

a) a=5;

c) a[3]++;

e) *(a+4)=3;

b) a++;

d) *a=1;

f) *a[3]=2;

21) Як розташовуються у пам'яті елементи матриць?

22) Навести усі способи виділення пам'яті під динамічну матрицю?

23) Пояснити використання двох зірочок поряд у команді:

`int **d = new int* [15];`

24) У програмі оголошено матрицю 3×6: `int *W=new int [3*6];`

Записати звернення до елемента з наведеними значеннями індексів i та j:

a) i=2, j=4

b) i=0, j=3

c) i=1, j=5

25) У програмі двовірний масив 4×5 оголошено так:

`double *a=new double [4*5];`

Записати значення індексів рядка та стовпчика при таких звертаннях до елементів цієї матриці:

a) *a

b) *(a+2)

c) *(a+15)

26) У програмі двовірний масив 4×5 оголошено так:

`double **A = new double * [4];`

`for(int i=0; i<4; i++) A[i]=new double [5];`

Записати індекси i та j елементів, звертання до яких у програмі має вигляд:

a) *(A[1]+2)

b) A[1][2]

c) *((A+1)+2)

27) Розтлумачити, чи є рівнозначними прототипи функцій:

`double fun(double (*z)[5], int n);`

`double fun(double z[][5], int n);`

28) У чому полягає відмінність роботи функцій `malloc()` та `calloc()`?

29) Яка функція застосовується для звільнення пам'яті, виділеної функцією `calloc()`?

30) Яким є призначення функції `realloc()`?

РОЗДІЛ 7

СИМВОЛИ І РЯДКИ

При виконуванні програм комп'ютер витрачає, за деякими оцінками, до 70% часу на маніпулювання з текстовими рядками: копіює їх з одного місця пам'яті в інше, перевіряє наявність у рядках певних слів, поєднує чи відсікає рядки тощо. У C++ є два основні види рядків: С-рядки (`char*`), які по суті є символьними масивами, і клас стандартної бібліотеки C++ `string`. Окрім того, Visual C++ надає свій доволі зручний в опрацюванні тип рядків `String`. Тому на початку розділу буде докладно описано особливості символьного типу `char`, потім буде розглянуто різновиди організації й опрацювання символьних масивів, які і є, власне, С-рядками, після чого буде наведено інформацію про `string`-рядки.

7.1 Символьний тип даних

Символами вважаються: великі й малі літери, цифри, знаки арифметичних дій ('+', '-', '*', '/', '='), пробіл, розділові знаки ('.', ',', ';', ':', '!', '?', '-'), службові символи, що відповідають клавішам [Enter], [Esc], [Tab] тощо. У C++ значення символьних констант записуються у одинарних лапках: '3', 'f', '+', '%'.

Для кодування усіх символів використовується восьмирозрядна послідовність 0 і 1, тобто один байт. Наприклад: символ цифри '9' кодується послідовністю бітів 0011 1001, символ літери латиниці 'w' – 0101 0111. За допомогою одного байта можна закодувати $2^8 = 256$ різних комбінацій бітів, а отже, 256 різних символів.

Щоб не було розходжень у кодуванні символів, існує єдиний міжнародний стандарт – так звана таблиця ASCII-кодів (American Standard Code for Information Interchange – американський стандартний код для обміну інформацією, див. додаток А). Символи ASCII мають коди від 0 до 127, тобто значення першої половини можливих значень байта, хоча часто кодами ASCII називають всю таблицю з 256 символів. Перші 128 ASCII-кодів є єдині для всіх країн, а коди від 128 до 255 називають розширеною частиною таблиці ASCII, де залежно від країни розташовується національний алфавіт і символи псевдографіки.

У таблиці ASCII всі символи пронумеровано, тобто вони мають власний унікальний код. Так само як у кожній мові людського спілкування існує алфавіт (перелік усіх літер у чітко визначеному порядку), усі комп'ютерні символи теж є суворо упорядкованими. Символ ' ' (пробіл) має код 32, цифри мають коди від 48 для '0' до 57 для '9', великі латинські літери – від 65 для 'A' до 90 для 'Z', малі літери латиниці – від 97 для 'a' до 122 для 'z'. Кодування другої половини таблиці ASCII має різні варіанти. Найпоширенішими є DOS-кодування (866 кодова сторінка), кодування 1251, яке є основним для Windows (див. додаток А),

кодування наборів символів Unicode (UTF-8, UTF-16, UTF-32 тощо)⁶. Звернімо увагу на різницю поміж цифрами і їхнім символьним зображенням. Наприклад, символ цифри '4' має ASCII-код 52 і не має безпосереднього відношення до числа 4.

Керувальні символи таблиці ASCII не мають символьного подання, тобто не мають візуального зображення, тому їх інколи називають недрукованими (non-printed), наприклад: [Esc], [Enter], [Tab] тощо. Ці символи розташовано в перших 32-х кодах таблиці ASCII-кодів. Звертатися до таких символів можна через їхній код чи за допомогою так званої ескейп-послідовності (escape).

Ескейп-послідовність – це спеціальна комбінація символів, яка розпочинається зі зворотної скісної риси і записується в одинарних лапках (табл. 7.1), наприклад: '\0', '\n'. Кожна з наведених у таблиці комбінацій символів вважається за один символ.

Таблиця 7.1

Деякі поширені у застосуванні ескейп-послідовності

Символьне подання	Опис
\n	символ переведення курсора на початок наступного рядка
\r	переведення каретки
\t	символ переведення курсора на наступну позицію табуляції (відповідає клавіші [Tab]),
\b	символ вилучення попереднього символу перед курсором (відповідає клавіші [BackSpace]),
\a	символ звукового сигналу системного динаміка
\\	символ \ (зворотна скісна риса)
\?	символ ? (знак запитання)
\'	символ ' (одинарні лапки)
\"	символ " (подвійні лапки)
\0	символ з кодом 0 є завершальним символом рядка символів
\0xВісімкове число	символ, код якого зазначено у вісімковій системі числення
\0xШістнадцяткове число	символ, код якого зазначено у шістнадцятковій системі числення

Тип символьних змінних у C++ називається **char**. Так, при оголошенні

```
char c, s, g;
```

в оперативній пам'яті для кожної з цих трьох змінних буде відведено по одному байту. Коли символьні змінні набувають певних значень, то комп'ютер зберігатиме в пам'яті не власне символи, а їхні коди. Наприклад, замість літери 'A' зберігатиметься її код 65. Тому, якщо присвоїти символьній змінній певне число, то C++ сприйме його як код символу з таблиці ASCII-кодів. Це поширюється лише на цілі числа. Зважаючи на різні кодування розширеної частини ASCII-таблиці для уникнення помилок вважається за оптимальне при роботі із симво-

⁶ Unicode 11.0 Character Code Charts. URL: <https://www.unicode.org/charts/#symbols> (data of reference: 29.09.2018).

лами використовувати їхнє символічне подання замість кодів. Наприклад, при оголошенні

```
char c = 115;
```

змінна `c` набуде значення символу `'s'`, який має код 115.

Окрім типу `char`, існує його беззнакова модифікація `unsigned char`. Дані типу `unsigned char` мають значення у діапазоні 0 ... 255. В ASCII-таблиці значення кодів літер кирилиці є більшими за 127, тому, якщо треба мати справу зі змінними, значеннями яких є літери кирилиці, їх слід оголошувати типом `unsigned char`:

```
unsigned char c;
```

Символи можна порівнювати. Більшим вважається той символ, у якого код є більший, тобто символ, розташований у таблиці ASCII-кодів пізніше. Наприклад: `'a' < 'h'`, `'A' < 'a'`.

Специфіка мови C++ при опрацюванні символічних даних полягає в тому, що за промовчанням тип є знаковим, тобто зберігає значення кодів символів від -128 до 127. При чому від'ємні значення мають коди розширеної частини таблиці ASCII. Наступний фрагмент програмного коду дозволить побачити як C++ інтерпретує символи кирилиці:

```
setlocale(LC_ALL, ".1251");
for (int i='A'; i<='я'; i++)
{ if (i%8==0) puts("\n");
  printf("%c = %3i ", (char)i, i);
}
```

А = -64	Б = -63	В = -62	Г = -61	Д = -60	Е = -59	Ж = -58	З = -57		
И = -56	Й = -55	К = -54	Л = -53	М = -52	Н = -51	О = -50	П = -49		
Р = -48	С = -47	Т = -46	У = -45	Ф = -44	Х = -43	Ц = -42	Ч = -41		
Ш = -40	Щ = -39	Ъ = -38	Ы = -37	Ь = -36	Э = -35	Ю = -34	Я = -33		
а = -32	б = -31	в = -30	г = -29	д = -28	е = -27	ж = -26	з = -25		
и = -24	й = -23	к = -22	л = -21	м = -20	н = -19	о = -18	п = -17		
р = -16	с = -15	т = -14	у = -13	ф = -12	х = -11	ц = -10	ч = -9		
ш = -8	щ = -7	ъ = -6	ы = -5	ь = -4	э = -3	ю = -2	я = -1		

Оскільки символічний тип `char` вважається у C++ за цілий тип, змінні цього типу можна додавати й віднімати. Результатом додавання буде символ, код якого дорівнює сумі кодів символів-доданків, наприклад:

```
char c = 'A';
char c1 = c + 5;    // c1 = 'F'
char c2 = c + 32;   // c2 = 'a'
char c3 = c - 10;   // c3 = '7'
```

У C++ існує ціла низка спеціальних функцій для роботи із символічними даними. Деякі з цих функцій наведено у табл. 7.2.

Таблиця 7.2

Функції для роботи із символами

Функція	Призначення
<code>tolower()</code>	повертає символ у нижньому регістрі
<code>toupper()</code>	повертає символ у верхньому регістрі
<i>Належність символу до множини перевіряють такі функції:</i>	
<code>isalnum()</code>	латинських літер та цифр ('A' – 'Z', 'a' – 'z', '0' – '9')
<code>isalpha()</code>	латинських літер ('A' – 'Z', 'a' – 'z')
<code>iscntrl()</code>	керувальних символів (з кодами 0...31 та 127)
<code>isdigit()</code>	цифр ('0' – '9')
<code>isgraph()</code>	видимих символів, тобто не є відповідним до клавіш [Esc], [Tab] тощо
<code>islower()</code>	латинських літер нижнього регістру ('a' – 'z')
<code>isprint()</code>	друкованих символів (<code>isgraph()</code> + пробіл)
<code>isupper()</code>	літер верхнього регістру ('A' – 'Z')
<code>ispunct()</code>	знаків пунктуації
<code>isspace()</code>	символів-роздільників

Примітка. Вищезазначені функції перевірки й перетворення регістру працюють лише з латинськими літерами. Спроба використати ці функції для символів кирилиці спричинить виведення повідомлення про помилку. Тому для роботи з літерами кирилиці слід використовувати перевірку належності символу до відповідного символного проміжку, а для перетворення регістру – зменшення чи то збільшення коду символу на різницю кодів між великими та малими літерами (для більшості літер вона становить 32).

Розглянемо деякі поширені алгоритми опрацювання символних змінних.

1) Щоб визначити код символу, треба значення цього символу присвоїти цілій змінній. І, навпаки, щоб дізнатися, який символ відповідає певному числу, слід це число присвоїти символіві. C++ сам виконає потрібні перетворення.

Наприклад, після виконання команд

```
int x; char c = 'n';
x = c;
```

змінна `x` набуде значення 110, яке відповідає коду символу 'n' в ASCII-таблиці.

2) Визначити, чи є символ `c` цифрою, можна двома способами: перевірити його належність до символного проміжку від '0' до '9' за допомогою умови:

```
if (c>='0' && c<='9') . . .
```

або застосувати функцію `isdigit()` для перевірки належності символу до множини цифр:

```
if (isdigit (c)) . . .
```

3) Дізнатися, чи є символ `c` великою латинською літерою, можна теж двома способами: перевірити його належність до символного проміжку від 'A' до 'Z' за допомогою умови:

```
if (c>='A' && c<='Z') . . .
```

або застосувати функцію `isupper()` для перевірки належності символу до множини великих латинських літер:

```
if (isupper (c)) . . .
```

4) Перевірку, чи є символ *c* латинською літерою, можна здійснити за допомогою умови

```
if (c>='A' && c<='Z' || c>='a' && c<='z') . . .
```

або за допомогою функції `isalpha()`:

```
if (isalpha (c)) . . .
```

5) Для перевірки, чи є символ *c* малою латинською літерою, слід записати умову

```
if (c>='a' && c<='z') . . .
```

або використати функцію `islower()` для перевірки належності символу до множини малих латинських літер:

```
if (islower(c)) . . .
```

6) Для перетворення малої латинської літери *c* на велику (верхній регістр) можна використати функцію `toupper()`:

```
c = toupper(c);
```

чи то відняти від значення літери різницю кодів між відповідними великими і малими літерами (вона становить 32):

```
char c = c - 32;
```

У аналогічний спосіб працює функція `tolower()`, яка збільшує код великих літер латиниці на 32 й, отже, здобуває *нижній регістр* літер латиниці.

Примітка. Вищезазначені функції перевірки й перетворення регістру працюють лише з латинськими літерами. Для виконання дій 1 – 5 з літерами кирилиці слід використовувати перевірку належності символу до відповідного символного проміжку, а для перетворення регістру – зменшення або збільшення коду символу на різницю кодів між великими і малими літерами (для більшості символів вона теж становить 32).

Для введення одного символу в консолі, крім вже знайомої потокової команди `cin>>`, існує функція `getch()` з бібліотеки `conio.h`.

```
char c = getch();  
getch();
```

Оскільки ця функція призупиняє виконання програми й очікує на введення символу, тобто на натиснення будь-якої клавіші, її доволі часто використовують наприкінці консольних програм перед `return 0`, щоб призупинити програму і надати користувачу можливість прочитати результати.

Приклад 7.1. Вивести всі символи та коди таблиці ASCII.

Текст програми:

```
#include <iostream>  
#include <locale>  
#include <conio.h>  
using namespace std;
```



```

int main()
{
    system("chcp 1251 > null");
    system("color F0");
    for (int i=32; i<256; i++)
    {
        if (i%4 == 0) printf("\n");
        printf("%3i = %c \t",i,(char)i);
    }
    getch();
    return 0;
}

```

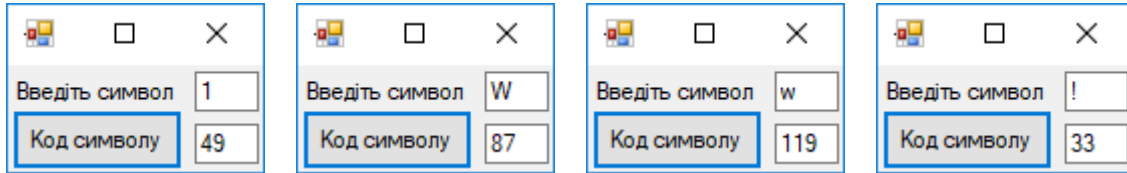
Результати роботи:

32 =	33 = !	34 = "	35 = #
36 = \$	37 = %	38 = &	39 = '
40 = (	41 =)	42 = *	43 = +
44 = ,	45 = -	46 = .	47 = /
48 = 0	49 = 1	50 = 2	51 = 3
52 = 4	53 = 5	54 = 6	55 = 7
56 = 8	57 = 9	58 = :	59 = ;
60 = <	61 = =	62 = >	63 = ?
64 = @	65 = A	66 = B	67 = C
68 = D	69 = E	70 = F	71 = G
72 = H	73 = I	74 = J	75 = K
76 = L	77 = M	78 = N	79 = O
80 = P	81 = Q	82 = R	83 = S
84 = T	85 = U	86 = V	87 = W
88 = X	89 = Y	90 = Z	91 = [
92 = \	93 =]	94 = ^	95 = _
96 = `	97 = a	98 = b	99 = c
100 = d	101 = e	102 = f	103 = g
104 = h	105 = i	106 = j	107 = k
108 = l	109 = m	110 = n	111 = o
112 = p	113 = q	114 = r	115 = s
116 = t	117 = u	118 = v	119 = w
120 = x	121 = y	122 = z	123 = {
124 =	125 = }	126 = ~	127 = ª
128 = º	129 = ¸	130 = ,	131 = ¸
132 = „	133 = ...	134 = ¸	135 = ¸
136 = €	137 = ¸	138 = ¸	139 = <
140 = ¸	141 = ¸	142 = ¸	143 = U
144 = ¸	145 = '	146 = '	147 = "
148 = "	149 = •	150 = -	151 = -
152 = ª	153 = ¸	154 = ¸	155 = >
156 = ¸	157 = ¸	158 = ¸	159 = u
160 =	161 = Ÿ	162 = Ÿ	163 = J
164 = ¸	165 = ¸	166 = ¸	167 = §
168 = ¸	169 = ¸	170 = ¸	171 = «
172 = ¸	173 = -	174 = ¸	175 = ¸
176 = ¸	177 = ±	178 = I	179 = i
180 = ¸	181 = µ	182 = ¸	183 = •
184 = ¸	185 = №	186 = ¸	187 = »
188 = j	189 = S	190 = s	191 = i
192 = A	193 = Б	194 = В	195 = Г
196 = Д	197 = Е	198 = Ж	199 = З
200 = И	201 = Й	202 = К	203 = Л
204 = М	205 = Н	206 = О	207 = П
208 = Р	209 = С	210 = Т	211 = У
212 = Ф	213 = Х	214 = Ц	215 = Ч
216 = Ш	217 = Щ	218 = Ъ	219 = Ы
220 = Ь	221 = Э	222 = Ю	223 = Я
224 = а	225 = б	226 = в	227 = г
228 = д	229 = е	230 = ж	231 = з
232 = и	233 = й	234 = к	235 = л
236 = м	237 = н	238 = о	239 = п
240 = р	241 = с	242 = т	243 = у
244 = ф	245 = х	246 = ц	247 = ч
248 = ш	249 = щ	250 = ъ	251 = ы
252 = ь	253 = э	254 = ю	255 = я

Приклад 7.2. Ввести один символ та визначити його код в таблиці ASCII.

Розв'язок. Для порівняння наведемо програмний код, реалізований з формою, та програму в консолі.

Вікно форми проекту з можливими результатами роботи програми:



Текст програми з формою:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    char c;           // Оголошення символної змінної для зберігання символу
    int k;            // та цілої змінної для зберігання його коду
    c=textBox1->Text[0]; // Введення значення символу
    k=c;              // Обчислення коду
    textBox2->Text=k.ToString();
}
```

Текст аналогічної програми у консолі:

```
#include <iostream.h>
int main()
{ char c;   int k;
  cout<<"Введіть символ: ";
  cin>>c;    // Введення значення символу
  k=c;
  cout<<"Код введенного символу: "<<k<<endl;
  cin.get();
  return 0;
}
```

7.2 Рядки

Символьні рядки можуть зберігати яку завгодно символну інформацію, приміром: імена файлів, назви книг, імена працівників та інші символні сполучення. Visual C++ підтримує кілька типів рядків – масиви символів, що перейшли від C (так звані C-рядки), клас `string` стандартної бібліотеки C++ та клас `String` середовища виконання CLR. Останні є доволі зручні у застосуванні, однак на практиці нерідкі є ситуації, коли виникає потреба у користуванні вбудованим типом (C-рядками).

Майже всі різновиди рядків у C являють собою послідовність (масив) символів із завершальним нуль-символом. Нуль-символ (нуль-термінатор) – це символ з кодом 0, який записується у вигляді керувальної послідовності `'\0'`. За розташуванням нуль-символу визначається фактична довжина рядка.

7.2.1 Char-рядки

Оголошення та специфіка організації char-рядків

У C++ рядки розглядають як масиви, елементи якого мають тип `char`. Рядок може містити символи літер, цифр і спеціальних символів, які записують у подвійних лапках. Ім'я рядка є константним вказівником на перший символ. Нумерація символів у рядку починається з 0. Останній символ рядка – службовий символ, так званий, завершальний нуль-символ `'\0'`, який є ознакою кінця рядка. При оголошенні рядка слід вказати максимальну кількість символів, які будуть зберігатися у рядку, при чому слід враховувати наявність наприкінці рядка нуль-термінатора і відводити додатковий байт для нього:

```
char s[10]; // Рядок s може містити до 9-ти символів
```

Рядок символів при оголошенні можна ініціалізувати початковим значенням, наприклад так:

```
char s[10] = "abcdef";
```

При цьому перші шість символів рядка набудуть значень кодів зазначених символів, а решта чотири символи – значення `'\0'`. До речі, якщо рядок при оголошенні ініціалізується, то зовсім не обов'язково вказувати його розмір в квадратних дужках:

```
char s[] = "abcdef";
```

При цьому розмір рядка визначається автоматично, а в кінець рядка дописується символ `'\0'`.

Відмінною рисою символьного масиву є те, що в ньому насправді може бути менше символів, ніж зазначено при оголошенні. Окрім того, з цими масивами можна виконувати певні специфічні дії, які не можна здійснювати з числовими масивами (наприклад, перевіряти наявність у масиві літери чи послідовності літер, копіювати масив як одне ціле, порівнювати масиви за алфавітом, дописувати один масив наприкінці іншого тощо).

Пам'ять під розміщення рядків, як і для будь-яких масивів, може виділятися як компілятором, так і динамічно – при виконуванні програми. Довжина динамічного рядка може задаватися змінною з визначеним заздалегідь значенням, а довжина статичного рядка має задаватися лише константою.

С-рядок може бути оголошеним в один з нижче наведених **способів**:

- 1) `char* s;` // Оголошення вказівника на перший символ рядка;
// пам'ять під сам рядок не виділяється.
- 2) `char ss[15];` // Оголошення рядка `ss` з 14 символів;
// пам'ять виділяється компілятором.
- 3) `const int n = 10;`
`char st[n];` // Оголошення рядка `st` з `n-1` (тобто 9) символів;
// пам'ять виділяється компілятором.
- 4) `int n = 10;`
`char* str = new char[n];` // Оголошення рядка `str` з `n-1` (тобто 9)
// символів; пам'ять виділяється динамічно.

При зазначенні довжини рядка слід враховувати завершальний нуль-символ. Так, у наведеному вище рядку `str` варто зберігати не 10, а лише 9 символів.

Зауважимо, що при оголошенні рядка першим способом пам'ять під рядок не виділяється і це може бути дуже небезпечним, оскільки у ту ж саму пам'ять можуть бути розміщені інші змінні і рядок буде втрачено.

При оголошенні рядок можна ініціювати рядковою константою, при цьому нуль-символ формується автоматично після останнього символу:

```
char str[10] = "Odessa";
```

При цьому виділиться пам'ять під масив з 9-ти елементів та 10-й – нуль-символ (всього 10 байт) і перші 6 символів рядка записуються в перші 6 байт цієї пам'яті (`str[0]='O', str[1]='d', str[2]='e', str[3]='s', str[4]='s', str[5]='a'`), а в сьомий елемент `str[6]` записується нуль-символ. Якщо рядок при оголошенні ініціалізується, його розмірність можна опускати (компілятор сам виділить потрібну кількість байтів):

```
char str[] = "Odessa"; // Виділено й заповнено 7 байтів
```

Рядки у лапках завжди неявно містять нуль-символ, тому при ініціалізації прописувати його немає потреби. Окрім того, різні наведені способи введення символьних масивів автоматично долучають нуль-символ у кінець масиву.

При оголошенні й ініціалізації масиву слід бути впевненим, що розмір масиву є достатній, щоб умістити всі символи рядка з нуль-символом. Річ у тім, що функції, які опрацьовують рядки, керуються позицією нуль-символу, а не розміром рядка. C++ не накладає жодних обмежень на довжину рядка.

Звернімо увагу на те, що рядкова константа (у подвійних лапках) і символьна константа (в одинарних лапках) не є взаємозамінними. Це константи різних типів. Символ у одинарних лапках, наприклад, `'s'` є *символьною* константою. Для зберігання такої константи компілятор C++ виділяє лише один байт пам'яті. Символ у подвійних лапках, наприклад, `"s"` є *рядковою* константою, що окрім символу `'s'` містить символ `'\0'`, який долучається компілятором. Більш того, `"s"` фактично являє собою адресу пам'яті, в якій зберігається рядок.

Як і числові масиви, символьні масиви опрацьовуються поелементно у циклі. Операція присвоювання одного рядка іншому є невизначена (оскільки рядок є масивом) і може виконуватися за допомогою циклу чи за допомогою функцій стандартної бібліотеки.

Поширеним засобом доступу до символів рядка є вказівники типу `char*`.

У прикладі

```
char *st = "Комп'ютерна програма";
```

компілятор записує всі символи рядка до масиву і присвоює змінній `st` адресу першого елемента масиву.

Рядок може вважатися за порожній у двох випадках: якщо вказівник на рядок має нульове значення `NULL` (немає взагалі жодного рядка) чи коли вказівник вказує на масив, який складається з одного нульового символу (не містить жодного значущого символу).

```
char *pc1 = 0; // pc1 не адресує жодного масиву символів,  
const char *pc2 = ""; // pc2 адресує нульовий символ
```

Введення-виведення char-рядків

Крім вже знайомих Вам потокових команд C++ `cin>>` та `cout<<` (з бібліотеки `iostream.h`), для введення й виведення рядків у консолі, використовують C-функції `gets/puts` та `scanf/printf` (з бібліотеки `stdio.h`).

1) Специфіка потокової команди введення `cin>>` полягає у тому, що вона вводить послідовність символів до першого пробілу або іншого розділювача, тобто цю команду недоречно використовувати для введення речень з пробілами.

2) Функція `puts(s)` виводить рядок `s` на екран, замінюючи нуль-символ на *Enter*.

3) Функція `gets(s)` зчитує символи з клавіатури до появи символу переведення рядка [Enter] і записує їх у рядок `s` (власне символ [Enter] до рядка не долучається, а замість нього записується нуль-символ). Функція `puts(s)` виводить рядок `s` на екран, замінюючи нуль-символ на [Enter], наприклад:

```
const int n=80;
char s[n];
gets(s);
puts(s);
```

Замість функції `gets()` вважається більш безпечним застосовувати функцію `gets_s()`, оскільки вона дозволяє контролювати кількість зчитуваних символів:

```
gets_s(s);
gets_s(s,80);
```

4) Для введення рядків у консолі можна використовувати метод `getline()`, який має два параметри: перший аргумент – рядок, який вводиться, а другий – кількість символів. Наприклад:

```
char s[20];
cout<<"Введіть рядок: "<<endl;
cin.getline(s, 20);
```

Рядок `s` у цьому фрагменті програми може прийняти лише 19 значущих символів і буде завершений нуль-термінатором (символом `'\0'`). Решту введених символів рядка буде проігноровано.

5) Функції форматного введення-виведення **`scanf`** та **`printf`** мають схожий формат і дозволяють вводити не лише рядки, а й числові дані, залежно від вказаного формату:

```
scanf (<формат>, <список_змінних>);
printf (<формат>, <список_змінних>);
```

де: *формат* – рядок специфікаторів формату у подвійних лапках. Найбільш поширеними специфікаторами є: `%s` – для рядка символів; `%i` – для цілих чисел; `%f` – для дійсних чисел у фіксованому форматі;
список_змінних – послідовність розділених комами змінних, значення яких вводиться або виводяться.

Так, функція `printf("Результат: ", str)` виводить рядок `str`, а функція `scanf("%s %i", str, &kol)` вводить значення рядка `str` і цілої змінної `kol`.

Встановлення мовних стандартів

Ще однією особливістю Visual C++ є те, що в консольному режимі різному інтерпретуються коди розширеної частини таблиці ASCII для введених символьних даних і набраного у програмі тексту. І саме тому текст кирилицею (українською або російською мовою) в консолі виводиться у вигляді абракадабри. Для визначення мовного стандарту (країна або регіон і мова) при роботі з розширеними символами існує спеціальна функція `setlocale()` з бібліотеки `locale.h`, формат якої має вигляд:

```
char *setlocale(int type, const char *locale);
```

де `type` – аргумент, який вказує на те, які саме дані мовного стандарту слід змінити: `LC_ALL` – усі; `LC_MONETARY` – формат грошових значень; `LC_NUMERIC` – формат числових значень з десятковою точкою або комою; `LC_TIME` – формат дати і часу тощо;

`locale` – аргумент, який задає мовний стандарт у вигляді рядка із зазначенням мови, коду країни і/або кодової сторінки у форматі: "<мова>_<країна>.<кодова сторінка>", при чому всі три складові зазначати не обов'язково. Наприклад, команди `setlocale(LC_ALL, ".1251");` `setlocale(0, "Ukrainian.1251");` `setlocale(0, "Ukrainian_Ukraine.1251");` і `setlocale(0, "uk-UA.1251");` для України є тотожними, а для США тотожними є використання команд `setlocale(0, "English");` `setlocale(LC_ALL, "English_United_States.1252");` і `setlocale(LC_ALL, "en-US");`, при чому перша з форм є найбільш рекомендованою.

Набір доступних імен мовних стандартів, назв мов, кодів країн і кодових сторінок можна побачити за інтернет-посиланням: <http://msdn.microsoft.com/ru-RU/goglobal/bb896001.aspx>. Приміром, функція `setlocale(0, "French_Canada.1252");` встановлює мовний стандарт "французький (Канада)" з кодовою сторінкою 1252.

Якщо в якості значення кодової сторінки аргументу `locale` вказати `".OCP"`, то встановлюватиметься кодова сторінка за промовчанням.

Оскільки мова C++ опрацьовує символ як один байт, то спроба встановлення кодової сторінки UTF-7 чи UTF-8, які потребують два байти для кожного символу, завершиться помилкою.

Функція `setlocale()` повертає вказівник на рядок, асоційований з параметром `type`. При виникненні помилки функція поверне нульовий вказівник.

Розглянемо на прикладі доцільність почергового використання різних кодувань (команди `setlocale(0, ".1251");` і `setlocale(0, ".OCP");`) у програмному коді для коректного виведення літер кирилиці, оскільки без використання цих команд кирилиця виводитиметься у вигляді абракадабри. Далі наведено три варіанти програмного коду, а праворуч від них – вигляд виведеного тексту:

```
1) char s[100];
   puts("Введіть рядок s =");
   gets_s(s);
   puts("Введений Вами рядок s =");
   puts(s);

2) char s[100];
   setlocale(LC_ALL, ".1251");
```



```
тїхфіЕН Ё фюь s =
Мова програмування C++
тїхфхэшц трьш Ё фюь s =
Мова програмування C++
```

```
puts("Введіть рядок s =");
gets_s(s);
puts("Введений Вами рядок s =");
puts(s);
```

```
Введіть рядок s =
Мова програмування C++
Введений Вами рядок s =
?Rÿ YаR?a гÿ --п C++
```

```
3) char s[100];
setlocale(LC_ALL, ".1251");
puts("Введіть рядок s =");
setlocale(LC_ALL, "Ukrainian_Ukraine.OCP");
gets_s(s);
setlocale(0, ".1251");
puts("Введений Вами рядок s =");
setlocale(0, ".OCP");
puts(s);
```

```
Введіть рядок s =
Мова програмування C++
Введений Вами рядок s =
Мова програмування C++
```

Отже, для виведення на екран текстових повідомлень з кирилицею слід використовувати кодування Windows 1251 (команда `setlocale(LC_ALL, ".1251")` або `setlocale(0, ".1251")`), а для виведення раніш введених символьних даних слід повернутися до початкових налаштувань командою `setlocale(0, ".OCP")`.

Функції стандартної бібліотеки для роботи з рядками

C++ має багату колекцію функцій опрацювання рядків із завершальним нулем. Якщо в рядку відсутній нуль-термінатор, опрацювання рядка може тривати скільки завгодно, допоки в пам'яті не зустрінеється `'\0'`. В якості аргументів до функцій переважно передаються вказівники на рядки. Якщо при виконванні функції здійснюється перенесення символів рядка з місця-джерела до місця-призначення, для рядка-призначення слід завчасно зарезервувати місце в пам'яті. Копіювання рядків з використанням просто вказівника, а не адреси початку завчасно оголошеного масиву – одна з найпоширеніших помилок програмування, навіть у досвідчених програмістів. При виділенні місця для рядка-призначення слід виділяти місце і для нуль-термінатора.

У табл. 7.3 наведено функції стандартної бібліотеки для роботи з C-рядками. Деякі з цих функцій, наприклад, `strlen()`, `strcpy()`, опрацьовують рядки, оголошені в будь-який з чотирьох раніш розглянутих способів. Але більшість функцій потребує, щоб рядок був оголошений як вказівник типу `char*` (див. способи 1 і 4 на початку п. 7.2.1). У консолі для використання цих функцій слід залучити до програми бібліотеку `<string.h>`. Перелік усіх функцій цієї та інших стандартних бібліотек C++ наведено у додатку В.

Таблиця 7.3

Функції стандартної бібліотеки `<string.h>`

Функція	Призначення	Формат
<code>strlen()</code>	повертає довжину рядка (без урахування символу завершення рядка)	<code>size_t strlen(char *s);</code>
<code>strcat()</code>	долучає s2 до s1 і, як результат, повертає s1	<code>char *strcat(char *s1, char *s2);</code>
<code>strncat()</code>	долучає до рядка s1 перші n символів з рядка s2	<code>char *strncat(char *s1, char *s2, size_t n);</code>

Продовження табл. 7.3

Функція	Призначення	Формат
<code>strlwr()</code>	перетворює всі латинські літери до нижнього регістру	<code>char *strlwr(char *s);</code>
<code>strupr()</code>	перетворює всі латинські літери до верхнього регістру	<code>char *strupr(char *s);</code>
<code>strcmp()</code>	порівнює рядки і повертає нульове значення, якщо рядки є однакові ($s1=s2$), від'ємне ($s1<s2$) чи додатне ($s1>s2$). Порівняння відбувається посимвольно і в якості результату повертається різниця кодів перших неоднакових символів	<code>int *strcmp(char *s1, char *s2);</code>
<code>strncmp()</code>	на відміну від попередньої функції, порівнює лише перші n символів рядка $s1$ з n символами рядка $s2$	<code>int *strncmp (char *s1, char *s2, size_t n);</code>
<code>stricmp()</code>	порівнює два рядки, не розрізняючи прописні й малі літери латиниці	<code>int stricmp (char *s1, char *s2);</code>
<code>strnicmp()</code>	порівнює лише перші n символів двох рядків, не розрізняючи великі й малі літери латиниці	<code>int strnicmp (char *s1, char *s2, size_t maxlen);</code>
<code>strcpy()</code>	копіює до $s1$ рядок $s2$, повертає $s1$, при цьому попереднє значення $s1$ втрачається	<code>char *strcpy (char *s1, char *s2);</code>
<code>strncpy()</code>	замінює перші n символів рядка $s1$ на перші n символів рядка $s2$	<code>char *strncpy (char *s1, char *s2, size_t n);</code>
<code>strchr()</code>	відшукує перше входження символу ch в рядок s і повертає вказівник на цей символ, тобто частину рядка s , розпочинаючи із символу ch і до кінця рядка. Якщо символу ch в рядку s немає, результат – <code>NULL</code>	<code>char *strchr(char *s, int ch);</code>
<code>strrchr()</code>	відшукує останнє входження символу в рядку і повертає частину рядка s , розпочинаючи з останнього входження символу ch і до кінця рядка. Якщо символу ch в рядку s немає, результат – <code>NULL</code>	<code>char *strrchr (char *s, int c);</code>
<code>strstr()</code>	відшукує підрядок $s2$ в рядку $s1$, повертає частину рядка $s1$, розпочинаючи з першого спільного символу для обох рядків і до кінця $s1$	<code>char *strstr(char *s1, char *s2);</code>
<code>strspn()</code>	повертає довжину початкового сегмента рядка $s1$, символи якого є в рядку $s2$	<code>size_t strcspn (char *s1, char *s2);</code>
<code>strcspn()</code>	повертає індекс першого символу в рядку $s1$, який є спільним для обох рядків (нумерація індексів символів розпочинається з нуля)	<code>size_t strcspn (char *s1, char *s2);</code>

Закінчення табл. 7.3

Функція	Призначення	Формат
strset()	замінює усі символи рядка s на символ c	char *strset(char *s, char c);
strnset()	замінює лише перші n символів рядка s на символ c	char *strnset (char *s, int ch, size_t n);
strpbrk()	відшукує місце першого входження у рядок s1 будь-якого символу рядка s2 і повертає частину рядка s1, розпочинаючи з цього символу і до кінця рядка	char *strpbrk (char *s1, const char *s2);
strrev()	реверс рядка s	char *strrev (char *s);
strtok()	повертає частину (лексему) рядка s1 від поточної позиції вказівника до розділового символу, зазначеного у рядку s2; зазвичай використовується для перетворення рядка на послідовність лексем	char *strtok(char *s1, const char *s2);

Розглянемо деякі з наведених у табл. 7.3 функцій детальніше на прикладах, для чого попередньо оголосимо вказівники на рядки s1, s2 та s3 і надамо їм початкові значення.

```
int k,n; char *s1="На Дерибасівській",*s2=" гарна погода",*s3;
n=strlen(s1);           // Обчислюється довжина рядка s1; n дорівнюватиме 17
k=strlen(s2);           // Обчислюється довжина рядка s2; k=13
k=strcmp(s1,s2);        // k=173; різниця між ASCII-кодами перших неоднакових
                        // символів рядків s1 і s2 дорівнює 173, а оскільки
                        // 173>0, можна стверджувати, що s1>s2

strcpy(s3,s2);          // s3=" гарна погода" (рядку s3 присвоюється рядок s2)
strncpy(s3,s1,2);       // Копіюються два перші символи з рядка s1 до s3
s3[2]='\0';             // Третім символом після символів"На" задається
                        // нуль-символ щоби обрізати рядок

strcat(s1,s2);          // s1="На Дерибасівській гарна погода"; до рядка s1
                        // долучається рядок s2

s3=strchr(s1, ' ');      // Пошук пробілів у рядку s1; тепер
                        // s3=" Дерибасівській
                        // гарна погода" – це рядок від першого пробілу до кінця
                        // рядка s1, тобто без першого слова

s3=strrchr(s1, ' ');     // Пошук останнього пробілу в рядку s1;
                        // тепер s3=" погода" – це останнє слово рядка s1

n=strchr(s1, ' ')-s1+1;  // n=3 – індекс першого пробілу в рядку s1, обчислений
                        // як різниця вказівників

k=strcspn(s1, " ") +1;  // k=3 – індекс першого пробілу в рядку s1; оскільки
                        // нумерація індексів символів розпочинається з 0, слід додати 1

s3=strstr(s1,s2);       // s3 = "гарна погода"; пошук рядка s2 у рядку s1
s3=strrev(s2);          // s3="адогос анраг" – відбувається реверс рядка
s3=strupr("Visual C++ "); // s3="VISUAL C++ " – перетворення усіх
                        // латинських літер рядка до верхнього регістру
```

```
s3=strlwr(s3);    // s3="visual c++" – зворотнє перетворення усіх
                  // латинських літер рядка до нижнього регістру
s3=strtok(s1, " "); // s3="На" – перше слово (лексема) рядка s1 до розділового
                  // знака, який зазначено символом (пробілом) у другому
                  // аргументі функції. Почергове здобуття усіх лексем
                  // рядка можна організовувати у циклі (див. приклад 7.16)
```

Отже, функції `strcpy()` та `strncpy()` призначено для копіювання рядка чи то його частини до іншого рядка. Функції `strchr()`, `strrchr()` та `strstr()` повертають вказівник на віднайдений символ чи підрядок.

Функція `strtok()` використовується для перетворювання рядка на послідовність лексем. **Лексема** – це послідовність символів, відокремлених символами-розділювачами (зазвичай пробілами чи знаками пунктуації). Наприклад, у рядку тексту кожне слово може розглядатися як лексема, а пробіли, що відокремлюють слова одне від одного, можна розглядати як розділювачі. Для того щоб розбити рядки на лексеми, треба організувати кілька викликів функції `strtok()` (за умови, що рядок вміщує понад одну лексему). Більш докладно розглянемо ці функції у наведених далі прикладах програм.

Створювання функцій опрацювання символьних масивів

Доволі часто при опрацюванні рядків доводиться здійснювати набори подібних команд для реалізації однакових дій, наприклад, для введення і виведення рядків, вилучення частини рядка тощо. У таких ситуаціях, а також для спрощення програмного коду і полегшення сприйняття програми, ці набори команд організовують як функції. Передавання рядків до функції є подібним до передавання масиву в якості параметра, при цьому треба зазначити адресу початку масиву (див. п. 5.2.4). Зробити це можна трьома способами:

```
double fun (int *a);
double fun (int a[ ]);
double fun (int a[25]);
```

За першого способу передається вказівник як посилання на масив, явно визначений в основній програмі. Більш докладно роботу з вказівниками розглянуто в розд. 6. У другій синтаксичній формі константний вираз у квадратних дужках є відсутній. Така форма може бути використана лише тоді, коли масив ініціалізується чи оголошується як формальний параметр. За третього способу явно зазначено кількість елементів масиву, що накладає обмеження на довжину опрацьовуваного рядка і, як наслідок, обмежує застосування цієї функції.

В якості прикладу організуємо функцію `show_string()` для виведення рядка на екран, параметром якої є рядок для виведення:

```
#include <iostream>
using namespace std;

void show_string(char s[])
{
    cout << s << endl;
}
```

```
int main()
{ show_string("Привіт, C++!");
  char *st="Я вчуса програмувати мовою C++");
  show_string(st);
  cin.get();
}
```

Оскільки нуль-символ свідчить про кінець рядка, функція `show_string()` не потребує додаткового параметра, який би задавав кількість символів у рядку.

Наведемо ще один приклад функції, яка обчислюватиме довжину рядка. І хоча є подібна бібліотечна функція, для кращого розуміння організації рядків не зайвим буде створення власної функції для реалізації цього алгоритму. Нижченаведена функція `length()` для визначення кількості символів шукає символ `'\0'` у рядку. Тип функції, тобто її результату є `int`, а оператор `return` повертає у якості результату довжину рядка, яка обчислюється як номер останнього символу рядка `'\0'`.

```
#include <iostream>
using namespace std;
int length(char *s)
{ int i;
  // У циклі перебираються усі символи, допоки не зустрінеться '\0'
  for (i=0; s[i] != '\0'; i++);
  return i; // Функція повертає довжину рядка, яка є номером нуль-символу
}
int main()
{ char s1[] = "Вчимося програмувати мовою C++";
  char *s2 = "Hello!";
  cout<<"Рядок "<<s1<<" містить "<<length(s1)<<" символів"<<endl;
  cout<<"Рядок "<<s2<<" містить "<<length(s2)<<" символів";
  cin.get();
}
```

Приклади програм з масивами символів

Приклад 7.3. Ввести рядок й вивести ASCII-коди його символів.

Розв'язок. Важливим аспектом, як було зазначено у попередньому підрозділі, є те, що друга половина кодів таблиці ASCII (яка зазвичай включає кирилицю) з одиничним старшим бітом, інтерпретується як від'ємні значення, оскільки тип `char` за стандартом мови може мати знакову реалізацію.

Текст консольної програми:

```
#include <iostream>
#include <locale>
using namespace std;
int main()
{ system("color F0");
  char s[20];
  setlocale(0, ".1251");
  puts("Введіть рядок s =");
```

```

setlocale(0, ".OCR");
gets_s(s);
setlocale(0, ".1251");
puts("Коди символів рядка:");
setlocale(0, ".OCR");
for (int i=0; i<strlen(s); i++)
    printf("%c = %3i ", (char)s[i], s[i]);
system ("pause>>void");
return 0;
}

```

Результати роботи:

```

Введіть рядок s =
Visual C++
Коди символів рядка:
V = 86 i = 105 s = 115 u = 117 a = 97 l = 108 = 32 C = 67 + = 43 + = 43

```

Приклад 7.4. Створити рядок з усіх великих літер англійської абетки.

Розв'язок. Оскільки тип `char` є порядковим типом, то його можна застосовувати у якості типу параметра циклу. Після застосування операції інкременту (`++`) змінна типу `char` набуде значення коду наступного символу.

Текст консольної програми:

```

#include <iostream>
using namespace std;
int main()
{ char s[27]; // 26 латинських літер і '\0'
  char c; int i=0;
  for(c = 'A'; c <= 'Z'; c++)
      s[i++] = c;
  s[i] = '\0'; // Записування символу '\0' до останнього символу рядка
  setlocale(0, ".1251");
  cout<< "Англійська абетка: " << s;
  cin.get();
  return 0;
}

```

Результат виконання програми:

Англійська абетка: ABCDEFGHIJKLMNOPQRSTUVWXYZ

Приклад 7.5. Ввести рядок і вивести замість цифр їх ASCII-коди.

Текст програми:

```

include <iostream>
#include <conio.h>
using namespace std;
int main()
{ char s[100];

```

```

puts("Input one string: ");
gets_s(s);
for (int i=0; i<strlen(s); i++)
    if (isdigit(s[i])) printf(" %i ",int(s[i]));
    else printf("%c",s[i]);
getch();
return 0;
}

```

Результати роботи:

Input one string:

digits: 0123456789

digits: 48 49 50 51 52 53 54 55 56 57

Приклад 7.6. Ввести рядок й обчислити кількість символів '*' у цьому рядку.

Розв'язок. Кількість зірочок обчислимо у функції `zirks`, яка отримає рядок `s`, а поверне до функції `main` ціле число – кількість зірочок у рядку.

Текст програми:

```

#include <iostream>
#include <locale>
#include <conio.h>
using namespace std;
int zirka(char* s)
{
    int i, k=0;
    for (i=0; s[i]!='\0'; i++)
        if (s[i]=='*') k++; // Якщо знайдено зірочку, збільшити кількість на 1
    return k;
}
int main()
{
    setlocale(LC_ALL, ".1251");
    char s[100]; int kol;
    puts("Введіть рядок: "); gets_s(s);
    kol = zirka(s); // Виклик функції
    printf("Кількість символів * у рядку = %i", kol);
    getch();
    return 0;
}

```

Результати роботи:

Введіть рядок:

128**Zz* *Я* *İİ

Кількість символів * у рядку = 6

Приклад 7.7. Ввести рядок і замінити всі крапки на символ '_' та обчислити кількість замін.

Текст консольної програми:

```
#include <iostream>
#include <locale>
int main()
{ system("color F0");
  char s[100]; int k=0;
  setlocale(0, ".1251"); puts("Введіть рядок s =");
  setlocale(0, ".OCR"); gets_s(s);
  setlocale(0, ".1251"); puts("Рядок після замін:");
  setlocale(0, ".OCR");
  for (int i=0; i<strlen(s); i++)
    if (s[i]=='.') {s[i]='_'; k++;}
  puts(s);
  setlocale(0, ".1251");
  printf("Кількість замін - %i", k);
  system("pause>>void");
  return 0;
}
```

```
Введіть рядок s =
я.ти.він.вона
Рядок після замін:
я_ти_він_вона
Кількість замін - 3
```

Приклад 7.8. Ввести рядок і визначити, чи є у ньому знаки пунктуації.

Текст програми:

```
#include <iostream>
#include <conio.h>
using namespace std;
int znaki(char* s)
{ for (int i=0; s[i]!='\0'; i++)
  if(s[i]=='.' || s[i]==',' || s[i]==';' || s[i]=='!' || s[i]=='?' || s[i]=='-')
    // або if(ispunct(s[i])) для латиниці
    return 1; //Якщо зустрічається знак пунктуації, функція поверне значення 1,
  return 0; //а якщо не зустрілося жодного, поверне значення 0
}
int main()
{ setlocale(LC_ALL, ".1251");
  char s[100]; int res;
  puts("Введіть рядок: "); gets_s(s);
  res = znaki(s);
  if (res) puts("Розділові знаки присутні");
  else puts("Розділових знаків немає");
  getch();
  return 0;
}
```

Результати роботи:

```
Введіть рядок:
Hello, world!
Розділові знаки присутні
```

```
Введіть рядок:
Несе Галя воду
Розділових знаків немає
```

Приклад 7.9. Ввести два рядки й визначити, чи є вони однаковими.

Текст програми:

```
#include <iostream>
#include <locale>
#include <conio.h>
using namespace std;
int perevirka(char* s1, char* s2)
{
    int i, n1, n2;
    n1=strlen(s1); n2=strlen(s2);
    if (n1 != n2) return 0; // Якщо довжини рядків не збігаються, повернути 0
    for (i=0; i<n1; i++) // Якщо знайшлись неоднакові символи,
        if(s1[i] != s2[i]) return 0; // повернути 0
    return 1; // Якщо однакових символів не знайдено, повернути 1
}
int main()
{ char s1[100], s2[100];
  setlocale(LC_ALL, ".1251");
  puts("Введіть перший рядок: "); gets_s(s1);
  puts("Введіть другий рядок: "); gets_s(s2);
  int res = perevirka(s1, s2);
  if(res) puts("Рядки однакові");
  else puts("Рядки не однакові ");
  getch();
  return 0;
}
```

Результати роботи:

Введіть перший рядок: 2+4	Введіть перший рядок: Hello	Введіть перший рядок: Програма
Введіть другий рядок: 4+2	Введіть другий рядок: Hello	Введіть другий рядок: програма
Рядки не однакові	Рядки однакові	Рядки не однакові

Приклад 7.10. Перевести до верхнього регістру (у великі літери) всі голосні літери латиниці рядка та обчислити їхню кількість.

```
include <iostream>
#include <conio.h>
using namespace std;
int lowreg(char* s)
{ int k=0;
  for (int i=0; i<strlen(s); i++)
      if (s[i]=='a' || s[i]=='o' || s[i]=='e' || s[i]=='i' || s[i]=='u')
          { s[i]-=32; k++; }
  return k;
}
```



```

int main()
{ char s[100];
  puts("Input one string: ");
  gets_s(s);
  int k=lowreg(s);
  puts(s);
  cout<<"The number of vowels - "<<k<<endl;
  getch();
  return 0;
}

```

```

Input one string:
Open the windows
OpEn thE wIndOws
The number of vowels - 4

```

Приклад 7.11. Видалити останнє слово з рядка.

```

include <iostream>
#include <locale>
#include <conio.h>
using namespace std;

void DeleteLastWord(char* s)
{ int i=0, ind=0;
  while (s[strlen(s)-1]==' ') s[strlen(s)-1]='\0'; // видалити кінцеві пробіли
  while (s[i]!='\0')
    if (s[i++]==' ') ind=i; // ind - номер останнього пробілу в рядку
  s[ind]='\0'; // записати замість пробілу символ кінця рядка
}

int main()
{ char s[100];
  setlocale(LC_ALL, ".1251"); puts ("Введіть рядок: ");
  gets_s(s);
  DeleteLastWord(s);
  puts("\nРядок з видаленням останнім словом:");
  setlocale(LC_ALL, ".ОСР");
  puts(s);
  getch();
  return 0;
}

```

```

Введіть рядок:
Малеча зростає швидко

Рядок з видаленням останнім словом:
Малеча зростає

```

Приклад 7.12. Ввести рядок і видалили з нього друге слово.

Розв'язок. У нижченаведеному кодї розв'язання організовано в окремій функції `del_word()`, яка видаляє з рядка `s`, починаючи з позиції `k`, `n` символів. У цій функції спочатку копіюється з рядка `s` перше слово з пробілом у допоміжний рядок `s1`. Після чого долучається до `s1` решта рядка `s` без слова, яке має бути видаленим (адреса початку цієї частини рядка обчислюється за допомогою адресної арифметики – зсув на `k+n+1` символів від початку рядка `s`).

Текст програми

```

#include <iostream>
using namespace std;

```

```

void del_word(char *s, int k, int n)
{ char *s1=new char [strlen(s)+1];
  strncpy(s1, s, k); s1[k]='\0';
  strcat(s1, s+k+n+1);
  strcpy(s,s1);
  delete [] s1;
}

int main()
{
  setlocale(LC_ALL, ".1251");
  int i, kol=0, p1=-1, p2=-1;
  char *s=new char [100];
  puts(" Введіть рядок:"); gets_s(s, 100);
  strcat(s, " ");
  for (i=0; s[i]!='\0'; i++)
    if (s[i]==' ')
      if (p1== -1) p1=i;
      else { p2=i; break; }
  del_word(s, p1+1, p2-p1-1);
  puts("\n Рядок без другого слова ");
  setlocale(0, ".OCP"); puts(s);
  delete []s;
  system("pause>>void");
  return 0;
}

```

Результат виконання програми:

```

Введіть рядок:
Microsoft developer network library

Рядок без другого слова
Microsoft network library

```

```

Введіть рядок:
Моя рідна країна

Рядок без другого слова
Моя країна

```

Приклад 7.13. Ввести рядок і видалили з нього слова, коротші 4-х символів.

Розв'язок. Для розв'язання задачі створимо новий рядок `s1` зі слів рядка `s`, довжина яких не менше 4-х символів.

Текст програми

```

#include <iostream>

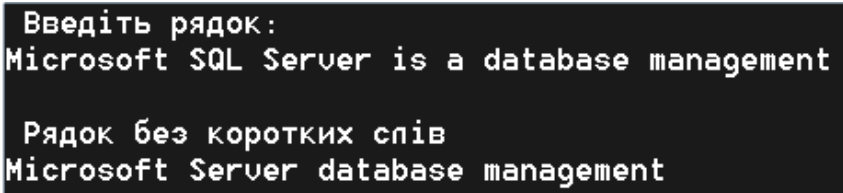
int main()
{ setlocale(LC_ALL, ".1251");
  char* s=new char [80], *t;
  char* s1=new char [80];
  strcpy(s1, "\0");
  puts(" Введіть рядок:"); gets_s(s, 80);
  t=strtok(s, " .,:?!-");
  while (t != NULL)
    { if (strlen(t)>=4) { strcat(s1,t); strcat(s1, " \0"); }

```

```

    t = strtok(NULL, " .,;?!-");
}
puts("\n Рядок без коротких слів ");
setlocale(0, ".ОСР");
strcpy(s, s1);
puts(s);
delete [] s;
delete [] s1;
system("pause>>void");
return 0;
}

```



Приклад 7.14. Ввести два рядки, визначити і вивести всі спільні для обох рядків символи та обчислити їхню кількість.

Текст програми:

```

include <iostream>
#include <locale>
#include <conio.h>
using namespace std;

int SameLetters(char* s1, char* s2, char* rez)
{ rez[0]='\0';
  int i, j, k, m, kol=0;
  for (i=0; s1[i]!='\0'; i++)
    for (j=0; s2[j]!='\0'; j++)
      if (s1[i] == s2[j])
        { for (k=m=0; rez[k]!='\0'; k++)
            if (s1[i]==rez[k])m=1;
          if (!m) { rez[kol]=s1[i]; rez[kol+1]='\0'; kol++;}
        }
  return kol;
}

int main()
{ char s1[100], s2[100],s3[100];
  setlocale(LC_ALL,".1251");   puts("Введіть перший рядок: ");
  setlocale(LC_ALL,".ОСР");    gets_s(s1);
  setlocale(LC_ALL,".1251");   puts("Введіть другий рядок: ");
  setlocale(LC_ALL,".ОСР");    gets_s(s2);
  int k = SameLetters(s1, s2, s3);
  setlocale(LC_ALL,".1251");
  if (!k) puts("\nНемає однакових символів");
  else printf("\nОднакових символів - %i\n", k);
  setlocale(LC_ALL,".ОСР");
  puts(s3);
  getch();
  return 0;
}

```

Результати роботи:

```
Введіть перший рядок:
мама мила раму
Введіть другий рядок:
зараз рама чиста

Однакових символів - 5
ма ир
```

```
Введіть перший рядок:
программер
Введіть другий рядок:
PROGRAMMER

Немає однакових символів
```

Приклад 7.15. Вивести ті літери латиниці (і малі, і великі), яких немає у рядку.

Текст програми:

```
#include <iostream>
#include <locale>
using namespace std;
int NotLetters(char* s, char *rez)
{ char c;
  int i,j,k,kol=0;
  for (c='A',i=0; c<='Z'; c++) rez[i++]=c;
  for (c='a'; c<='z'; c++) rez[i++]=c;
  rez[i]='\0';
  for (i=0; s[i]!='\0'; i++)
    if (s[i]>='A' && s[i]<='Z' || s[i]>='a' && s[i]<='z' )
      for (j=0; rez[j]!='\0'; j++)
        if (s[i] == rez[j])
          { kol++;
            for (k=j; rez[k]!='\0'; k++) rez[k]=rez[k+1];
          }
  return kol;
}
int main()
{ char s1[100], s2[55];
  setlocale(LC_ALL, ".1251");
  puts("Введіть рядок: ");    gets_s(s1);
  int k = NotLetters(s1,s2);
  if (!k) puts("\nУ рядку немає літер латиниці:");
  else printf("\nУ рядку %i літер(и) латиниці (без урахування регістру),
             \nale серед них немає таких:\n", k);

  puts(s2);
  system("pause>>void");
  return 0;
}
```

Результати роботи:

```
Введіть рядок:
Голосні літери латиниці такі: aoueіAOUeI

У рядку 10 літер(и) латиниці (без урахування регістру),
але серед них немає таких:
BCDFGHJKLMNPQRSTUWXYZbcd fghjklmnpqrstvwxyz
```

```
Введіть рядок:
Ходить котик тихо, буде мишкам лихо...

У рядку немає літер латиниці:
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
```

Приклад 7.16. Ввести рядок і вивести окремо у стовпець усі слова цього рядка.

Розв'язок. Наведемо чотири способи розв'язання цього завдання у чотирьох різних функціях `words`. Вигляд основної програми буде однаковим, тому наведемо її тільки один раз у першому способі.

У першому способі слова виводитимуться разом із розділовими знаками, якщо такі будуть. Тут перебираються по черзі усі символи і формуються слова із символів до чергового пробілу.

Перший спосіб:

```
#include <iostream>
#include <locale>
#include <conio.h>
using namespace std;

void words(char* s)
{ char sl[100];
  int k=0, i, n;
  strcat(s, " "); // Долучити пробіл у кінець рядка
  n=strlen(s);    // Довжина рядка
  if (n<2) return; // Якщо рядок був порожній, вийти з функції
  sl[0]='\0';     // Слово "порожнє"
  for (i=0; i<n; i++) // Цикл для перебору символів рядка
  { if (s[i] != ' ') // Якщо символ не є пробілом, тобто слово не закінчилось,
    { sl[k]=s[i];    // записати символ у слово
      sl[k+1]='\0';  // і сформувати кінець рядка
      k++;
    }
    else // Інакше, коли символ є пробілом,
    { if (strlen(sl)>0) puts(sl); // вивести слово та
      sl[0]='\0';
      k=0; // "обнулити" слово перед формуванням нового
    }
  }
}

int main()
{
  char s[100];
  setlocale(LC_ALL, ".1251"); puts("Введіть рядок: ");
  setlocale(LC_ALL, ".ОСР");  gets_s(s);
  words(s);
  getch();
  return 0;
}
```

Результати роботи:

Введіть рядок: A NULL pointer indicates an error A NULL pointer indicates an error	Введіть рядок: Мама мила раму. Тепер рама чиста. Мама мила раму. Тепер рама чиста.

Другий спосіб:

Для розбивання рядка на слова (лексеми) застосовуватимемо функцію `strtok()`, яка має два аргументи: рядок, який треба розбити на лексеми, і рядок, який містить символи-розділювачі.

Перший виклик функції `strtok` у програмі

```
t = strtok(s, " .,;?!-");
```

присвоїть змінній `t` вказівник на перше слово рядка `s`. Другий аргумент наведеної функції містить можливі розділові символи. Функція `strtok()` відшукує перший символ у рядку `s`, який не є розділювачем. Це є початок першого слова. Потім функція віднаходить розділювач у рядку і замінює його на нуль-символ (`'\0'`). Цим завершується поточне слово. Функція `strtok()` зберігає вказівник на наступний символ рядка `s` за даним словом і повертає вказівник на поточну лексему (слово). У наступних викликах `strtok()` у програмі для почергового виокремлення слів з рядка `s` першим аргументом функції зазначається `NULL`, для того, щоб виокремлення наступного слова рядка `s` відбувалося з позиції, яку було збережено попереднім викликом `strtok()`. Якщо лексем при виклику `strtok()` більше немає, `strtok()` повертає `NULL` – і відбувається вихід із циклу.

Зверніть увагу, що `strtok()` модифікує вхідний рядок; тому, якщо рядок після виклику `strtok()` знову використовуватиметься у програмі, треба завчасно зробити копію рядка.

Цей алгоритм можна застосовувати для виокремлення чисел, але тоді в якості розділових знаків не слід застосовувати крапку, яка може інтерпретуватися як десяткова крапка.

Нагадаємо, текст основної програми є таким самим, як і в першому способі.

Текст функції:

```
void words(char* s)
{ char *t;
  t=strtok(s, " .,;?!-");
  while (t != NULL)
  { puts(t);
    t = strtok(NULL, " .,;?!-");
  }
}
```

Введіть рядок: Hi, Tony! How are you? - I'm OK, thanks. Hi Tony How are you I'm OK thanks

Третій спосіб:

Для виокремлення слів рядка вважатимемо за ознаку завершення слова пробіл. Оскільки після останнього слова рядка здебільшого пробіл є відсутній, тому долучаємо його до рядка `s` у заздалегідь зарезервоване місце.

Рухаючись рядком, почергово виокремлюємо частину рядка `s` у змінну `slovo`. Виокремлення слова можна реалізувати за допомогою функції `strtok()` чи то, як у нижченаведеному тексті програми, функції `strncpy()`. Ця функція копіює частину рядка `s`, розпочинаючи з поточної позиції, а кількість літер копіювання визначається позицією пробілу. Після цього лишилося долучити до слова завершальний нуль і вивести слово окремим рядком.

Текст функції:

```
void words(char* s)
{
    strcat(s, " ");
    int i, n=strlen(s);
    char *slovo = new char[n+1];
    while (strchr(s, ' ')) // Допоки у рядку є пробіли,
    {
        for (i=0; i<n+2 && s[i]!=' '; i++); // відшукується позиція пробілу і
        strncpy(slovo, s, i); // копіюються символи до пробілу у змінну slovo
        slovo[i]='\0';
        puts(slovo);
        s+=i+1; // Перехід до наступного слова
    }
    delete[]slovo;
}
```

Четвертий спосіб:

Іноді при виокремленні слів рядка виникає потреба у зберіганні цих слів в однотипному масиві для їхнього подальшого опрацювання. Для виділення пам'яті для масиву `arr` вказівників на слова спочатку слід визначити кількість слів (`kol`). Зверніть увагу, що в програмі ознака завершення слова визначається як: `i`-й символ не є пробілом (`!isspace(s[i])`), а `(i+1)`-й символ – пробіл (`isspace(s[i+1])`).

Текст функції:

```
void words(char* s)
{
    int n=strlen(s), i, j=0, kol=0;
    char *slovo=new char[n+1];
    strcat(s, " ");
    for (i=0; i<n-1; i++)
        if (!isspace(s[i]) && isspace(s[i+1])) kol++;
    char **arr=new char *[kol];
```



```

while (strchr(s, ' '))
{ for (i=0; i<n+2 && s[i]!=' '; i++);
  strncpy(slovo, s, i);
  slovo[i]='\0';
  arr[j]=new char[strlen(slovo)+1];
  strcpy(arr[j], slovo);
  j++;
  s += i+1;
}
s -= n;
for(i=0; i<kol; i++) puts(arr[i]);
for(i=0; i<kol; i++) delete []arr[i];
delete []arr;
delete []slovo;
}

```

Приклад 7.17. Вивести індекси всіх входжень літери 'a' до рядка.

Розв'язок. Для виконання завдання доцільно використати функцію `strchr()`, яка відшукує перше входження відповідного символу до рядка і повертає вказівник на цей символ, тобто частину рядка `s`, розпочинаючи з цього символу і до кінця рядка. Якщо символу, зазначеного другим аргументом функції, в рядку немає, результатом функції є `NULL`.

Текст програми у консолі:

```

#include <iostream>
using namespace std;
int main ()
{
    char s[80], *p;
    setlocale(0, ".1251"); puts("Введіть рядок:");
    setlocale(0, ".OCP"); gets(s);
    setlocale(0, ".1251"); cout<<"Відшукуємо літеру 'a' у рядку \";
    setlocale(0, ".OCP"); cout<<s<<"\":" <<endl;
    setlocale(0, ".1251");
    p=strchr(s, 'a');
    if (p==NULL)
    { puts("Не знайдено!");
      system("pause>>void");
      return 0;
    }
    while(p!=0)
    { printf ("Знайдено у позиції %d\n", p-s+1);
      p=strchr(p+1, 'a');
    }
    system("pause>>void");
    return 0;
}

```

Результат виконання програми:

Введіть рядок:

I am happy!

Відшукуємо літеру 'a' у рядку "I am happy!":

Найдено у позиції 3

Найдено у позиції 7

Введіть рядок:

I'm hungry!

Відшукуємо літеру 'a' у рядку "I'm hungry!":

Не найдено!

Приклад 7.18. Ввести рядок і перевірити, чи є він символьним поданням цілого числа.

Розв'язок. Перевірку, чи є символ цифрою, можна здійснити чи то за допомогою функції `isdigit()`, чи за перевірки належності його до діапазону цифр.

Текст програми у консолі:

```
#include <iostream>
int main ()
{
    char st[20];
    setlocale(0, ".1251");
    printf("Введіть ціле число: ");
    scanf("%s", &st);
    int i, n=strlen(st);
    if ((st[0]<'0' || st[0]>'9') && st[0]!='-')
    { printf("Рядок не є цілим числом");
      system("pause>>void");
      return 0;
    }
    for (i=1; i<n; i++)
        if (st[i]<'0' || st[i]>'9')
        { printf("Рядок не є цілим числом");
          system("pause>>void");
          return 0;
        }
    printf("Рядок є цілим числом.\n");
    system("pause>>void");
    return 0;
}
```

Результати виконання програми:

Введіть ціле число 23.5

Введений рядок не є цілим числом.

Введіть ціле число -56

Введений рядок є цілим числом.

Приклад 7.19. Ввести рядок і перевірити, чи є він дійсним числом.

Текст програми у консолі:

```
#include <iostream>
int main ()
{
    char st[20];
    int err=0; // Спочатку вважаємо, що рядок не є дійсним числом
    setlocale(0, ".1251");
    printf("Введіть дійсне число: ");
    scanf("%s",&st);
    int i, k=0, n=strlen(st);
    if (!isdigit(st[0])&&st[0]!='-') err=1; // Перший символ – цифра чи '-'?
    for (i=1; i<n-1; i++)
    {
        if (!isdigit(st[i]) && st[i]!='.') // Якщо символ не є цифрою чи
        { err=1; break; } // десятковою крапкою, переривається перевірка решти
        if (isdigit(st[i-1]) && st[i]=='.' && isdigit(st[i+1]))
        { k++; //Якщо символ є десятковою крапкою, а попередній і наступний
          if (k>1) // символи – цифри, перевіряється можливість повторів таких
          { err=1; break; } // ситуацій і, якщо '.' вже була, перевірка припиняється
        }
    }
    if (k==0) err=1; // Десяткових крапок усередині цифр не було?
    if (err==1) printf("Не є дійсним числом!");
    else printf("Є дійсним числом!");
    system("pause>>void");
    return 0;
}
```

Результати виконання програми:

```
Введіть дійсне число: qwerty
Не є дійсним числом!
Введіть дійсне число: 5.6
Є дійсним числом!
```

Приклад 7.20. Створити програму для введення рядка і можливості вилучення із заданої позиції певної кількості символів рядка. Значення кількості символів для вилучення й номера позиції ввести з клавіатури.

Розв'язок. Цей алгоритм є поширеним у застосуванні і розв'язувати його можна по-різному, наприклад, за допомогою допоміжного рядка і вибирання до нього символів початкового рядка без тих символів, які треба «вилучити», після чого слід перезаписати створений рядок до початкового. Наведений нижче алгоритм вилучення частини рядка є більш зручним, оскільки не потребує використання допоміжного рядка. Розглянемо цей алгоритм на прикладі рядка s:

```
char *s="Погода дуже тепла";
```

У розглянутому нижче прикладі ціла змінна `pos` є номером символу, з позиції якого слід розпочати вилучення, `kol` – кількість символів, що їх буде вилучено, `&s[pos]` – частина рядка `s`, розпочинаючи із символу з індексом `pos`, а `pos+kol` – індекс першого символу, який не буде вилучено. Якщо `pos=7` і `kol=5`, команда

```
strcpy(&s[pos], &s[pos+kol]);
```

переставить (скопіює) слово «тепла» з нуль-символом на місце слова «дуже» унаслідок присвоювання вказівників. При цьому перша частина рядка залишиться на місці. Отже, слово «дуже» з пробілом зникне (його буде вилучено), а рядок стане коротшим.

s						&s[pos]						&s[pos+kol]					
П	о	г	о	д	а		д	у	ж	е		т	е	п	л	а	\0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
П	о	г	о	д	а		т	е	п	л	а	\0					

Текст програми у консолі:

```
#include <iostream>
using namespace std;
void del(char*s, int pos, int kol)
{ pos--; // Коригування номера, оскільки нумерація символів розпочинається з 0
  if (!(&s[pos+kol]==NULL || &s[pos]==NULL || pos<0 || kol<0))
    strcpy(&s[pos], &s[pos+kol]);
}
int main ()
{ int pos, kol; char *s=new char[80];
  setlocale(0, ".1251"); printf("Введіть рядок: ");
  setlocale(0, ".OCP"); gets(s);
  cout<<"pos = "; cin>>pos;
  cout<<"kol= "; cin>>kol;
  del(s,pos,kol);
  setlocale(0, ".1251"); printf("Рядок після видалення: ");
  setlocale(0, ".OCP"); puts(s);
  system("pause>>void");
  return 0;
}
```

Результати виконання програми:

```
Введіть рядок: Погода дуже тепла!
pos = 7
kol= 5
Рядок після видалення: Погода тепла!
```

Приклад 7.21. Ввести рядок і на його базі створити новий рядок з однолітерних слів.

Текст програми у консолі:

```
#include <iostream>
void onchar(char*s1, char*s2)
{ int N=strlen(s1);
  strcpy(s2, "");
  char *D=" .,?!-"; // Розділювачі
  char *p=new char [N];
  p=strtok(s1,D);    // Читання першого слова
  while (p!=0)       // Допоки не кінець рядка,
  { if (strlen(p)==1) // перевіряється розмір слів і однолітерні слова
    { strcat(s2,p); // долучаються до рядка s2
      strcat(s2," "); // разом з пробілом
    }
    p=strtok(NULL,D); // Виокремлення наступного слова
  }
  delete []p;
}
int main ()
{ int pos, kol;
  char *s=new char[80], *rez=new char [80];
  setlocale(0, ".1251"); printf("Введіть рядок: ");
  setlocale(0, ".OCP");  gets(s);
  onchar(s,rez);
  setlocale(0, ".1251"); printf("Однолітерні слова: ");
  setlocale(0, ".OCP");  puts(rez);
  system("pause>>void");
  return 0;
}
```

Результати виконання програми:

Введіть рядок: Я, ти і вона з ним!
Однолітерні слова: Я і з

Приклад 7.22. Ввести рядок і вивести всі слова, які розпочинаються і закінчуються на однакову літеру.

Розв'язок. Для виокремлення слів рядка вважатимемо ознакою кінця слова пробіл. Оскільки після останнього слова здебільшого пробіл є відсутній, тому долучаємо його до рядка *s* у заздалегідь зарезервоване місце.

Рухаючись рядком, по чергово виокремлюємо слова і запам'ятовуємо у змінній *p* індекс першої літери (початку) поточного слова. Виокремлення слова можна реалізувати за допомогою функції *strtok()* або, як у наведеному нижче тексті програми, функції *strncpy()*. Ця функція копіює частину рядка *s* до рядка *slovo*, розпочинаючи з індексу *p*. Кількість літер, яку треба скопіювати, тобто довжину слова можна задати різницею індексів віднайденого пробілу і першого

символу `i`-р. Після цього неодмінно слід долучити до слова завершальний нуль. Якщо у слові перша і остання літери збігаються, відбудеться виведення слова окремим рядком.

Текст програми у консолі:

```
#include <iostream>
int main ()
{ char *s=new char[80];
  setlocale(0, ".1251"); printf("Введіть рядок: ");
  setlocale(0, ".OCP");  gets(s);
  strcat(s, " ");
  int i, n=strlen(s), p=0;
  char *slovo = new char[n+1];
  setlocale(0, ".1251");
  printf("Слова, які починаються і завершуються однаковою літерою:\n");
  setlocale(0, ".OCP");
  for (i=0; i<n+2; i++)
  { if (s[i]==' ')          // Якщо віднайдено пробіл,
    { strncpy(slovo, &s[p], i-p); // відбувається копіювання слова
      slovo[i-p]='\0';
      if (s[p]==s[i-1]) // Перша й остання літери слова збігаються?
        puts(slovo);
      p=i+1;           // Запам'ятовування початку нового слова
    } }
  delete[]s;
  delete[]slovo;
  system("pause>>void");
  return 0;
}
```

Результати виконання програми:

Введіть рядок: Над морем розкинулась арка веселки

Слова, які починаються і завершуються однаковою літерою:

морем

арка

Приклад 7.23. Ввести рядок і на його базі створити новий рядок зі слів, які містять цифри.

Текст програми:

```
#include <iostream>
int main ()
{
  char *s=new char[80]; // Виділення пам'яті для введеного рядка s,
  setlocale(0, ".1251"); printf("Введіть рядок: ");
  setlocale(0, ".OCP");  gets(s);
  strcat(s, " ");
}
```

```

int i, n=strlen(s), p=0;
char* slovo=new char[n+1]; // Виділення пам'яті для слова
char* rez=new char[n+2];   // і результуючого рядка rez
strcpy(rez,"");
setlocale(0, ".1251");
printf("Слова, які містять цифри:\n");
setlocale(0, ".ОСР");
for (i=0; i<n+2; i++)
{ if (s[i]==' ')
  { strncpy(slovo, &s[p], i-p);
    slovo[i-p]='\0';
    for (int j=0; j<strlen(slovo); j++)// Перебираються усі символи слова
      if (slovo[j]>='0' && slovo[j]<='9')// і, якщо у ньому віднайдено цифру,
      { strcat(rez,slovo);           // то це слово приєднується до rez
        strcat(rez, " ");           // разом із пробілом
        break;                     // Припинення подальшої перевірки цього слова
      }
    p=i+1;
  }
}
puts(rez);
delete[]s; delete[]slovo; delete[]rez;
system("pause>>void");
return 0;
}

```

Результати виконання програми:

Введіть рядок: qw ss1 234 1ww я23м уф

Слова, які містять цифри:

ss1 234 1ww я23м

Приклад 7.24. Ввести рядок й визначити довжину першого й останнього слів у рядку (розділювачем слів вважати пробіл).

Розв'язок. Для визначення довжини першого слова слід знайти індекс першого пробілу. Для визначення довжини останнього слова треба знайти індекс останнього пробілу, обчислити різницю довжини рядка та цього індексу і зменшити її на 1.

Текст програми:

```

#include <iostream>
#include <string.h>
#include <clocale>
using namespace std;

int main()
{
    char *s=new char [100];

```



```

setlocale(LC_ALL, ".1251"); puts(" Введіть рядок:");
gets_s(s, 100);
int pp=0, op=0, n=strlen(s), i;
for (i=0; i<n; i++)
    if (s[i]==' ') // Зупинитись, якщо зустріли перший пробіл
        { pp=i; cout << "\n Довжина першого слова: " << pp << endl; break;}
if (!pp) {cout<<"\nПробілів немає"<<endl; cin.get(); return 0;}
for (i=n-1; i>=0; i--)
    if (s[i]==' ') { op=i; break;} // Зупинитись, якщо зустріли пробіл
cout<< "\n Довжина останнього слова: " << n-op-1 << endl;
delete []s;
cin.get();
return 0;
}

```

Результат виконання програми:

Введіть рядок: програма	Введіть рядок: Одеська національна академія зв'язку – super
Пробілів немає	Довжина першого слова: 7

Приклад 7.25. Ввести рядок і слово, яке слід вставити після першого слова рядка.

Розв'язок. Алгоритм, який вставляє до рядка *s* слово *slovo* на позицію *k*, реалізуємо у функції. Позицію для вставляння слова, яка є індексом символу наступного після першого пробілу рядка, задамо як різницю вказівників на пробіл і на початок рядка, збільшену на одиницю: `strchr(s, ' ')-s+1`.

Функція `strcat()` долучає свій другий аргумент до першого аргументу, при цьому перший символ другого рядка записується замість нульового символу (`'\0'`), який завершував перший рядок. Програміст має бути впевненим, що перший рядок, є достатньо великий для того, щоб зберегти комбінацію першого рядка, другого рядка та нуль-символу, який копіюється з другого рядка.

Текст програми:

```

#include <iostream>
void str_insert(char *s, char *slovo, int k)
{ // Долучення до рядка slovo частини рядка s, розпочинаючи з позиції k
  strcat(slovo, &s[k]);
  // Копіювання в рядок s, розпочинаючи з позиції k, рядка slovo
  strcpy(&s[k], slovo);
}
int main ()
{ char *s=new char[100]; // Виділення пам'яті для рядка s
  setlocale(0, ".1251"); printf("Введіть рядок: ");
  setlocale(0, ".OCP"); gets(s); // Введення рядка
  strcat(s, " ");
}

```

```

int i, n=strlen(s);
char* slovo=new char[n+1]; // Виділення пам'яті для слова
setlocale(0, ".1251"); printf("Слово для вставки: ");
setlocale(0, ".ОСР"); gets(slovo); // Введення слова
strcat(slovo, " "); // Долучення пробілу до слова
setlocale(0, ".1251"); printf("Результат: ");
// Якщо в рядку s немає пробілу, він долучається
if (!strchr(s, ' ')) strcat(s, " ");
int pos=strchr(s, ' ')-s+1; // Індекс першого пробілу рядка
str_insert(s, slovo, pos); // Вставлення слова в рядок у позицію pos
setlocale(0, ".ОСР"); puts(s); // Виведення рядка
system("pause>>void");
return 0;
}

```

Результати виконання програми:

Введіть рядок: Сьогодні гарна погода

Слово для вставки: дуже

Результат: Сьогодні дуже гарна погода

Приклад 7.26. Для введенного рядка поміняти місцями вказані номерами слова.

Текст програми

```

#include <iostream>

int main()
{ setlocale(LC_ALL, ".1251");
  char s[100] = {'\0'}, rez[100] = {'\0'}, *word;
  char swap[25], a[25][20]; // для переставлення слів
  int k = -1, i, l, r;
  printf("Введіть рядок:\n");
  gets(s);
  word = strtok(s, " \\\"'-! ,");
  while (word)
  { k++; // кількість слів
    strcpy(a[k], word);
    word = strtok(NULL, " \\\"'-! ,");
  }
  printf("Всього %d слів(ова)\n", k+1);
  printf("\nВведіть номери слів у рядку для обміну: ");
  if (scanf("%d %d", &l, &r)==0 || l>r || r>k+1)
  { printf("\nНеправильне введення номерів слів для обміну!!!\n");
    return 1;
  }
  strcpy(swap, a[l-1]); // переставлення слів у масиві слів
  strcpy(a[l-1], a[r-1]);
  strcpy(a[r-1], swap);
  for (i=0; i<=k; i++)

```

```

{   strcat(rez,a[i]);           // поєднання слів у рядок
    strcat(rez, " ");
}
puts("\nПісля обміну слів:");
setlocale(LC_ALL, ".ОСР");
puts(rez);
system("pause>>void");
return 0;
}

```

```

Введіть рядок:
Нехай все буде добре
Всього 4 слів(ова)

Введіть номери слів у рядку для обміну: 3 4

Після обміну слів:
Нехай все добре буде

```

Приклад 7.27. На основі введеного рядка створити новий рядок з маленьких голосних літер латиниці, які містяться у введеному рядку.

Розв'язок. Спочатку треба обчислити кількість маленьких голосних літер латиниці у введеному рядку *s*, щоб виділити необхідну кількість пам'яті для символів створюваного рядка *s1*. Після цього можна копіювати у новий рядок *s1* голосні маленькі літери латиниці з рядка *s*, а наприкінці долучити завершальний нуль до нового рядка.

Текст програми:

```

#include <iostream>
int main ()
{
    setlocale(LC_ALL, ".1251");
    int i, n, kol=0, j=0;    char *s=new char [100];
    puts(" Введіть рядок:"); gets_s(s, 100);
    n=strlen(s);
    for (i=0; i<n; i++)
        if (s[i]=='a' || s[i]=='e' || s[i]=='i' || s[i]=='o' || s[i]=='u') kol++;
    char *s1 = new char [kol+1];
    for (i=0; i<n; i++)
        if (s[i]=='a' || s[i]=='e' || s[i]=='i' || s[i]=='o' || s[i]=='u')
            { s1[j]=s[i]; j++; }
    s1[kol]='\0';
    puts("\n Новий рядок з голосних літер ");
    setlocale(0, ".ОСР");    puts (s1);
    delete [] s;
    system("pause>>void");
    return 0;
}

```

Результат виконання програми:

```

Введіть рядок:
Visual Studio Integrated Development Environment

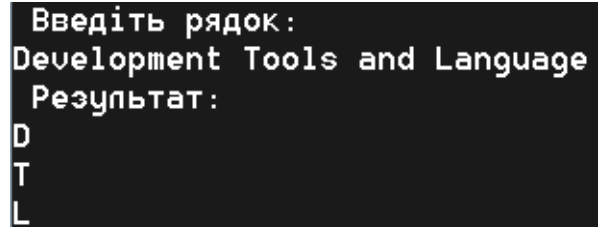
Новий рядок з голосних літер
iauaioeaeaeioeioe

```

Приклад 7.28. Ввести рядок і вивести всі заголовні літери, які стоять на початку слів. Слова розділяються одним чи кількома пробілами.

Текст програми:

```
#include <iostream>
int main()
{
    setlocale(LC_ALL, ".1251");
    int i, n, kol=0, j=0;
    char *s=new char [100];
    puts(" Введіть рядок:");
    gets_s(s, 100);
    n=strlen(s);
    puts(" Результат:");
    setlocale(0, ".ОСР");
    if (s[0]!=' ' && isupper(s[0])) // Перше слово перевіряємо окремо
        cout<<s[0]<<endl;
    for (i=0; i<n-1; i++)
        if (s[i]==' ' && s[i+1]!=' ' && isupper(s[i+1]))
            cout<<s[i+1]<<endl;
    delete [] s;
    system("pause>>void");
    return 0;
}
```



Функції перетворювання рядка char* на число

Виконати перетворювання рядка на число можна багатьма способами. Перший – використання бібліотечних функцій `atoi()`, `atof()`, `atol()`. Ці функції входять до стандартної бібліотеки мови C і мають такий формат:

```
int atoi(const char* str);
long atol(const char* str);
double atof(const char* str);
```

Результатами перетворювання цих функцій є числа відповідного типу. Функції цілочисельного перетворювання `atoi()` і `atol()` сприймають такий формат перетворюваного рядка:

[пробіли][знак]цифри

а функція перетворювання рядка в дійсне число `atof()`, відповідно:

[пробіли][знак][цифри][.цифри][{d | D | e | E}[знак]цифри]

Тут пробіли – будь-який зі знаків пробілу, табуляції ('`\t`'), вертикальної табуляції ('`\v`') – при перетворюванні ігноруються. Знак – символ '+' чи '-' (якщо його не зазначено, то вважається, що число є додатне). Цифри – символи від '0' до '9'. Для числа з рухомою крапкою, якщо немає цифр до символу '.', має бути хоча б одна цифра після нього. Дійсне число може бути подано в експоненціальній формі з одним із символів-префіксів експоненти.

Основний недолік цих функцій полягає в тому, що вони ніяк не сигналізують про помилку, якщо така станеться в перебігу перетворювання рядка на число, наприклад, через невідповідність його формату чи то з інших причин. Цю проблему розв'язують такі стандартні бібліотечні функції перетворювання рядків на числа:

```
long strtol(const char* str, char **endptr, int radix)
unsigned long strtoul(const char* str, char* *endptr, int radix)
double strtod(const char* str, char* *endptr)
```

Ці функції мають такі відмінності від попередньої групи:

- ✓ через параметр `endptr` вони повертають вказівник на перший символ, який не може бути інтерпретований як частина числа;
- ✓ відбувається контроль переповнення і, якщо це сталося, функції сигналізують встановленням змінної `errno` у значення `ERANGE`, а також повертають, відповідно, `LONG_MAX/LONG_MIN`, `ULONG_MAX/ULONG_MIN` та `+/-HUGE_VAL` залежно від знака числа у переданому рядку;
- ✓ функція `strtod()` використовує інформацію про поточні встановлені через `setlocale` регіональні налагодження, а, отже, може коректно інтерпретувати числа із символом `'.'` як розділювачем цілої та дробової частини;
- ✓ для функцій `strtol()` і `strtoul()` можна задавати параметром `radix` основу системи числення, при цьому, якщо `radix` дорівнює 0, основа визначається автоматично за першими символами числа.

Типове використання цих функцій може мати вигляд:

```
char* end_ptr;
long val = strtol(str, &end_ptr, 10);
if (*end_ptr)
{ . . . // Сигналізування про помилку в рядку
}
if ((val == LONG_MAX || val == LONG_MIN) && errno == ERANGE)
{ . . . // Сигналізування про переповнення
}
// Продовження штатної роботи
```

Перетворення числа на рядок

Для перетворення цілого числа на рядок можна скористатися такими функціями:

```
char* itoa(int value, char* string, int radix);
char* ltoa(long value, char* string, int radix);
char* ultoa(unsigned long value, char* string, int radix);
```

Для цих функцій перший вхідний параметр `value` – ціле число, яке треба перетворити на рядок; другий параметр `string` є буфером, до якого буде записано результат перетворювання, а третій параметр `radix` – це основа системи числення, в якій буде подано число.

Приклад 7.29. Ввести рядок – математичний вираз, що міститиме два числа і знак арифметичної дії поміж ними та обчислити значення цього виразу.

Текст програми у консолі:

```
#include <iostream>
using namespace std;
int main ()
{ char *s=new char [80];
  int n,i; double x,y,z;   char c;
  gets(s);
  n=strlen(s);
  x=atof(s);
  for (i=1; i<n; i++)
    if (!isdigit(s[i])&& s[i]!='.')
      if (s[i]!='+' && s[i]!='-' && s[i]!='*' && s[i]!='/')
        { cout<<"Error"; cin.get(); return 0;}
      else { c=s[i]; y=atof(&s[i+1]); break;}
  switch (c)
  { case '+': z=x+y; break;
    case '-': z=x-y; break;
    case '*': z=x*y; break;
    case '/': z=x/y; break;
  }
  cout << x << c << y << "=" << z << endl;
  system("pause>>void");
  return 0;
}
```

Результати роботи програми:

```
23,5+6
Error
23.5-7.6
23.5-7.6=15.9
```

7.2.2 Рядки string

Строго кажучи, у мові C ++ відсутній вбудований рядковий тип; у типах `char` і `wchar_t` зберігаються окремі символи. Для імітації рядка (їх ще називають «рядки у стилі C») треба оголосити масив цих типів, додавши кінцеве значення `NULL` (наприклад, ASCII `'\0'`) до елемента масиву за останнім символом. Рядки у стилі C вимагають написання набагато більшого обсягу коду або використання зовнішніх бібліотек службових функцій. Однак у сучасному C++ є стандартні бібліотечні типи `std::string` (для 8-розрядних символьних рядків типу `char`) або `std::wstring` (для 16-розрядних символьних рядків типу `wchar_t`). Ці контейнери STL можна розглядати як власні рядкові типи, оскільки вони є частиною стандартних бібліотек, що є у будь-якому сумісному середовищі розроблення C++. Для цього слід просто скористатися директивою `#include <string>`, щоб ці типи стали доступними у програмі.

У сучасній мові C++ настійно не рекомендується використовувати масиви символів з завершальним значенням NULL (рядки у стилі C, робота над якими розглядається у п. 7.2.1)⁷.

Клас рядків `string` стандартної бібліотеки C++ дозволяє працювати з рядками як зі звичайними типами: копіювати, присвоювати та порівнювати рядки як базові типи. Щоб використовувати рядки класу `string`, треба залучити до програми відповідний заголовний файл:

```
#include <string>
```

а також написати після цього оператор (пояснення див. підрозд. 9.4):

```
using namespace std;
```

Оголошення рядків `string` може бути таким:

```
string s;
string st("рядок символів\n"); // Оголошення рядка з ініціалізацією
string s(st);                  // Ініціалізування рядка s змінною st типу string
```

Більш докладно способи оголошення й ініціалізації рядків розглянуто далі у прикладах 7.30 ... 7.37.

Порівнювання двох рядків виконується за допомогою операцій відношення: `==`, `!=`, `<`, `<=`, `>`, `>=`. Присвоювання (копіювання) рядків виконується за допомогою звичайного присвоювання, а конкатенація (сполучання, зчеплення) рядків виконується за допомогою операції «+».

```
s = st;                // Рядок st присвоюється рядкові s
string s2 = s + st;    // Долучення рядків
s = s + st;            // Долучення рядка st у кінець рядка s
```

Операція конкатенації рядків може сполучати рядки `string` не лише між собою, але й з C-рядками, наприклад:

```
string s1 ("hello");
const char *pc=", ";
string s2 ("world");
string s3 = s1 + pc + s2 + "\\0";
```

Цей вираз працює завдяки тому, що компілятор «знає», в який спосіб можна автоматично перетворювати C-рядки та рядки `string`. Можливе й просте присвоювання C-рядка рядковій `string`:

```
string s1;                // Оголошення порожнього рядка
const char *pc = "символьний масив";
s1 = pc;
```

Зворотне перетворення не працює. Спроба зробити наступну ініціалізацію C-рядка спричинить помилку компіляції:

```
char *str = s1;          // Помилка компіляції
```

Щоб здійснити таке перетворення, слід застосувати функцію `c_str()`, яка повертає вказівник на константний масив:

```
const char *str = s1.c_str();
```

До окремих символів рядків `string` можна звертатися за допомогою індексу.

⁷ Тип системы C++ (современный C++) // Microsoft. Developer Network. URL: <https://msdn.microsoft.com/ru-ru/library/hh279663.aspx> (дата звернення 7.10.2018).

Нумерація символів розпочинається з 0.

Найбільш поширені у застосовуванні функції опрацювання string-рядків наведено у табл. 7.4.

Таблиця 7.4

Деякі функції і методи опрацювання рядків string

Назва	Призначення	Формат
append()	долучає рядок s у кінець рядка	string& append (string& s);
assign()	замінює значення рядка на значення іншого рядка s	string& assign (string& s);
copy()	копіює n символів рядка, починаючи з позиції pos, у символьний масив s	size_t copy (char *s, size_t n, size_t pos = 0);
empty()	повертає true, якщо рядок є порожнім. Приклад перевірки, чи є рядок порожнім: <code>if (s.empty()) . . .</code>	bool empty ();
erase()	вилучає частину рядка довжиною n символів, розпочинаючи з позиції p. Існує декілька форматів запису цієї функції	string erase (size_t p, size_t n);
find()	відшукує і повертає позицію першого входження підрядка str в рядок. Функція може мати два параметри, тоді в якості першого зазначається підрядок str, в якості другого – позиція pos, розпочинаючи з якої слід здійснювати пошук	size_t find (string &str); size_t find (char* str, size_t pos);
insert()	вставляє слово str у рядок на позицію p	string insert (size_t p, const string& str);
length()	обчислює довжину рядка, наприклад: <code>int k=s.length();</code>	size_t length ();
replace()	замінює len елементів рядка, розпочинаючи з індексу idx, на символи рядка str. Існує кілька форматів запису цього методу	replace (size_t idx, size_t len, string& str);
reserve()	змінює рядок у такий спосіб, що всі його символи записуються у зворотному порядку, наприклад: <code>s.reserve();</code>	void reserve ();
resize()	змінює розмір рядка на n. Якщо новий розмір є менше за довжину рядка, рядок усікається	void resize (size_t n);
size()	обчислює довжину рядка, наприклад: <code>int k=s.size();</code>	size_t size ();
substr()	повертає частину рядка розміром n символів (чи то менше, якщо символів не вистачає), яка розпочинається з індексу pos	string substr (size_t pos, size_t n);
swap()	переставляє значення рядка і аргумента s, наприклад: <code>s1.swap(s2);</code>	void swap (string &s);

Розглянемо приклади деяких найпоширеніших операцій з рядками.

У рядку `str` всі крапки на знаки підкреслювання змінює наступний фрагмент програми

```
string str("fa.disney.com");
int n = str.size();
for (int i=0; i<n; i++)
    if (str[i] == '.') str[i] = '_';
```

Заміну крапок на знаки підкреслювання у рядку можна записати однією командою за допомогою функції `replace()`:

```
replace(str.begin(), str.end(), '.', '_');
```

`str.begin()` та `str.end()` – вказівники відповідно на початок і кінець рядка `str`.

Функція `substring()` копіює частину рядка. Наприклад:

```
string s1 = str.substr(n);
```

копіює у рядок `s1` з рядка `str` всі символи з позиції `n` до кінця, а команда

```
string s2 = str.substr(n,12);
```

копіює у рядок `s2` з рядка `str` 12 символів з позиції `n`.

Функція `resize(n)` встановлює довжину рядка. Якщо `n<=s.size()`, то «зайві» символи відкидаються.

Приклади програм з рядками `string`

Приклад 7.30. Ініціалізувати `string`-рядки у різні способи.

Текст програми у консолі:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{ setlocale(0, ".1251");
  system("color F0");
  char *line="short line for testing";
  string s1="Program"; // Звичайне ініціалізування
  cout << "s1 = " << s1 << endl;
  string s2(s1); // Ініціалізування string-рядка; s2="Program"
  cout << "s2 = " << s2 << endl;
  string s3(line); // Ініціалізування C-рядком; s3="short line for testing"
  cout << "s3 = " << s3 << endl;
  // Ініціалізування частиною C-рядка (з початку рядка):
  // перший аргумент – C-рядок, другий – кількість символів;
  string s4(line,10); // s4="short line"
  cout << "s4 = " << s4 << endl;
  // Ініціалізування частиною C-рядка (з середини рядка): перший аргумент –
  // C-рядок, другий – початкова позиція, третій – кількість символів.
  string s5(s3,6,4); // Копіювання слова "line" з рядка s3;
  cout << "s5 = " << s5 << endl; // s5="line"
  // Ініціалізування рядком зазначених символів:
```

```

// перший аргумент – кількість символів, другий – символ для ініціалізування;
string s6(15, '*');          // s6 = "*****"
cout << "s6 = " << s6 << endl;
// Ініціалізування частиною string-рядка (з середини рядка): перший аргумент –
// вказівник на початок частини, другий – вказівник на кінець частини;
string s7(s3.begin(), s3.end()-5);    //s7 = "short line for te"
cout << "s7 = " << s7 << endl;
// Можна ініціалізувати рядок присвоюванням:
string s8 = "Program";             //s8 = "short line for te"
cout << "s8 = " << s8 << endl;
cin.get();
return 0;
}

```

Результат роботи програми:

```

s1 = Program
s2 = Program
s3 = short line for testing
s4 = short line
s5 = line
s6 = *****
s7 = short line for te
s8 = Program

```

Приклад 7.31. Демонстрація роботи функції `append()`.

Текст програми у консолі:

```

#include <iostream>
#include <string>
using namespace std;
int main()
{ string str = "Visual C++";
  string s   = ""; // Порожній рядок
  char *ch   = "abcdef";
  // У кінець рядка s долучаються перші (з позиції 0) шість символів рядка str;
  s.append(str, 0, 6);          // s="Visual"
  cout << "s = " << s << endl;
  // У кінець рядка s долучаються символи рядка str з позиції 6 до кінця;
  s.append(str.begin()+6, str.end()); // s="Visual C++"
  cout << "s = " << s << endl;
  // У кінець рядка s долучаються три знаки оклику
  s.append(3, '!');             //s= "Visual C++!!!"
  cout << "s = " << s << endl;
  // У кінець рядка s долучаються перші три символи C-рядка ch
  s.append(ch, 3);              //s= "Visual C++!!!abc"
  cout << "s = " << s << endl;
  cin.get(); return 0;
}

```

Результати виконання програми:

```
s = Visual
s = Visual C++
s = Visual C++!!!
s = Visual C++!!!abc
```

Приклад 7.32. Демонстрація роботи функції `assign()`.

Текст програми у консолі:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{ string s = "", str = "Visual C++";
  char *ch = "Character array";
  s.assign(str);      // Рядку s присвоюється рядок str
  cout << "s = " << s << endl;
  // Рядку s присвоюються 3 символи рядка str, розпочинаючи з позиції 7;
  s.assign(str,7,3);   // s = "C++".
  cout << "s = " << s << endl;
  // Рядку s присвоюється C-рядок ch;
  s.assign(ch);        // s = " Character array "
  cout << "s = " << s << endl;
  // Рядку s присвоюються перші 9 символів C-рядка ch;
  s.assign(ch,9);      // s = " Character"
  cout << "s = " << s << endl;
  // Рядку s присвоюється частина рядка str з позиції str.begin() до str.end();
  s.assign(str.begin(),str.end()); // s = "Visual C++"
  cout << "s = " << s << endl;
  // Рядку s присвоюється рядок з 17-ти зірочок;
  s.assign(17, '*');    // s = "*****"
  cout << "s = " << s << endl;
  cin.get();  return 0;
}
```

Результати виконання програми:

```
s is : Visual C++
s is : C++
s is : Character array
s is : Character
s is : Visual C++
s is : *****
```

Приклад 7.33. Демонстрація роботи функцій `find()` та `copy()`. Програма шукає символ ':' і копіює у символьний масив початок рядка `str` до віднайденного символу включно.

Текст програми у консолі:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{ string str = "It is 8:30 now"; char ch[255];
  cout << "str = " << str << endl;
  int n = str.find(':'); // Визначення позиції символу ':' у рядку str
  // У символьний масив ch копіюється n+1 символ з початку рядка str
  str.copy(ch, n+1, 0);
  ch[n+1] = 0; // У кінець символьного масиву долучається '\0'
  cout << "ch = " << ch << endl;
  cin.get(); return 0;
}
```

Результат виконання програми:

```
str = It is 8:30 now
ch = It is 8:
```

Приклад 7.34. Демонстрація роботи функцій `empty()` та `erase()`. Рядок `str` складається з 7-ми зірочок. Допоки цей рядок є непорожній, з нього послідовно вилучається останній символ.

Текст програми у консолі:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
  string str = "*****";
  while ( ! str.empty() )
  { cout << str << endl;
    str.erase(str.end()-1);
  }
  cout << endl;
  cin.get();
  return 0;
}
```

Результат виконання програми:

```
*****
*****
*****
****
***
**
*
```

Приклад 7.35. Два рядки заповнюються символами англійської абетки. Демонструється робота функції `erase()`.

Текст програми у консолі:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{ string str, s;
  for ( char ch = 'a'; ch <= 'z'; ch++ )
    str.append(1,ch);
  s = str;
  cout << "str = " << str << endl;
  cout << "s   = " << str << endl<<endl;
  // Вилучається 13 символів від початку рядка str
  str.erase(0,13);
  cout << " str = " << str << endl;
  // Вилучається 13 символів рядка s, починаючи з 14-го
  str = s.erase(13,13);
  cout << " s   = " << str << endl<<endl;
  // Вилучається другий символ рядка s
  s.erase(s.begin()+1);
  cout << "s     = " << s << endl;
  // З рядка s вилучаються символи з першого до четвертого
  s.erase(s.begin(),s.begin()+4);
  cout << "s       = " << s << endl;
  cin.get();
  return 0;
}
```

Результати виконання програми:

```
str = abcdefghijklmnopqrstuvwxyz
s   = abcdefghijklmnopqrstuvwxyz
str = nopqrstuvwxyz
s   = abcdefghijklm
s   = acdefghijklm
s   = fghijklm
```

Приклад 7.36. Демонструється робота функції `find()`.

Текст програми:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
  setlocale(0, ".1251");
  string str("C++ is best language");
```

```

cout << "str = " << str << endl;
int pos = str.find("best"); // Шукається позиція слова "best" у рядку str
cout << "Позиція слова best = " << pos << endl;
// Шукається друге входження "best" у рядок, якщо не знайдено, повертається -1
pos = str.find("best", pos+1);
cout << "Друге входження best = " << pos << endl;
pos = str.find('g'); // Шукається перше входження символу 'g'
cout << "Позиція символу 'g' = " << pos << endl;
string s = "is";
pos = str.find(s); // Шукається перше входження слова «is» (string-рядок)
cout << "Позиція входження слова is = " << pos+1 << endl;
cin.get();
return 0;
}

```

Результати виконання програми:

```

str = C++ is best language
Позиція слова best = 7
Друге входження best = -1
Позиція символу 'g' = 15
Позиція входження слова is = 5

```

Приклад 7.37. Демонструється робота функції insert().

Текст програми:

```

#include <iostream>
#include <string>
using namespace std;
int main()
{
    setlocale(0, ".1251");
    string str="C++ language";
    string s="is best";
    char ch[]="Java language";
    cout << "str = " << str << endl;
    cout << "s = " << s << endl;
    cout << "ch = " << ch << endl;
    int pos=4;
    str.insert(pos,s); // у рядок str на позицію pos
    cout << "str після вставлення рядка s: "<<str<<endl;
    int n = str.find('l');
    // Вставляння пробілу в рядок str на позицію str.begin() + n
    str.insert(str.begin() + n, ' ');
    cout << "str після вставлення пробілу: "<<str<<endl;
    // Вставляння у кінець рядка str трьох знаків оклику
    str.insert(str.end(),3,'!');
    cout << "str після вставлення 3-х знаків оклику: "<<str<<endl;
    // Вставляння на початок рядка s чотирьох символів із C-рядка ch

```



```

s.insert(0,ch,5);
cout << "s після вставлення 5-ти символів з ch: "<<s<<endl;
// Вставлення 9 символів з рядка str, починаючи з позиції n-1 у рядок s на позицію X
n = str.find('l');
int X = s.length();
s.insert(X,str,n-1,9);
cout << "s після вставлення 9-ти символів з str: "<<s<<endl;
// Вставлення у рядок s частини рядка str з begin() до begin()+7
// на позицію begin()+5
s.insert(s.begin()+5,str.begin(),str.begin()+4);
cout << "s після вставлення частини рядка: "<<s<<endl;
cin.get();
return 0;
}

```

Результати виконання програми:

```

str = C++ language
s =
ch = Java language
str після вставлення рядка s: C++ is bestlanguage
str після вставлення пробілу: C++ is best language
str після вставлення 3-х знаків оклику: C++ is best language!!!
s після вставлення 5-ти символів з ch: Java is best
s після вставлення 9-ти символів з str: Java is best language
s після вставлення частини рядка: Java C++ is best language

```

7.2.3 Клас String

Оголошення та специфіка організації рядків класу String

Компілятор Visual C++ підтримує рядки, які є об'єктами і подають текст як послідовність символів Unicode. Visual C++ підтримує рядкові змінні, значення яких є неявними, і літерали, значенням яких є явний рядок у лапках.

Середовище виконання Windows і середовище CLR подають рядки String як об'єкти, пам'ять під які виділяється і керується автоматично. Це означає, що не потрібно явно звільняти пам'ять для рядка, якщо змінна виходить за межі області видимості або коли застосунок припиняє роботу. Для оголошення автоматично керованих рядків слід зазначити тип String з модифікатором дескриптор-об'єкта (^):

```
String ^<змінна>;
```

Наприклад:

```
String ^s, ^str = "abcdefg";
```

При такому оголошенні друга зі змінних не лише оголошується, а й ініціалізується початковим значенням. Ініціалізація оголошуваного рядка значенням, введеним з клавіатури, може мати вигляд:

```
String ^myString = Console::ReadLine();
```

Для визначення довжини рядка можна скористатися властивістю Length:

```
int n = str->Length;
```

Максимальний розмір об'єкта String – 2 ГБ пам'яті, або близько 1 млрд. символів.

Рядки класу String складаються із символів типу Char, тобто символів Unicode з кодуванням UTF-16 (16-бітове значення з діапазону шістнадцяткових значень 0x0000 – 0xFFFF).

Звернення до кожного символу рядка виконується за допомогою індексу, як до елемента масиву:

```
myString [i]
```

Конкатенація (зчеплення) рядків виконується за допомогою оператора +:

```
String ^s = "This is one sentence. " + "This is a second. ";  
s += "This is a third sentence.";  
Console.WriteLine(s);
```

У результаті цього сформується і виведеться на екран рядок:

```
"This is one sentence. This is a second. This is a third sentence."
```

Доволі зручним є введення та виведення рядків String за допомогою елементів керування на формі, оскільки відповідні текстові поля (властивості) елементів мають цей тип. Наприклад, введення та виведення рядка S через елемент textBox1 можна подати в такий спосіб:

```
S = textBox1 ->Text;           // Введення рядка з textBox1  
textBox2 ->Text = S;           // Виведення рядка до textBox2
```

Деякі методи для роботи з рядками⁸

1) **IndexOf** – пошук підрядка у рядку. Метод повертає індекс (нумерація від нуля) першого входження підрядка або зазначеного символу Unicode в межах даного екземпляра. Метод повертає -1, якщо знак або підрядок не знайдено в даному екземплярі. Пошук виконується з місця, вказаного у другому параметрі. Якщо другий параметр відсутній, то пошук розпочинається з початку рядка.

Наприклад, команда для визначення номера першого пробілу в рядку s матиме вигляд:

```
int n = s->IndexOf(" ");
```

Метод IndexOf, як і більшість методів для роботи з класом String, мають декілька форм: з одним, двома і, навіть, трьома аргументами. З докладнішою інформацією можна ознайомитись за інтернет-адресою:

[http://msdn.microsoft.com/ru-ru/library/system.string.indexof\(v=vs.110\).aspx](http://msdn.microsoft.com/ru-ru/library/system.string.indexof(v=vs.110).aspx).

2) **Substring** – отримання підрядка з рядка, тобто копіювання частини рядка, починаючи з певної позиції і до кінця рядка:

```
String^ Substring(int startIndex)
```

Друга форма цього методу дозволяє задавати певну довжину копійованої частини рядка:

```
String^ Substring(int startIndex, int length)
```

Наприклад, виокремлення другого слова рядка s можна здійснити так:

```
String ^s = "This sentence has five words." , ^word2;
```

⁸ Ці та інші методи, конструктори, оператори та властивості класу можна дослідити за посиланням: <https://msdn.microsoft.com/ru-ru/library/system.string.aspx?cs-save-lang=1&cs-lang=cpp#code-snippet-1>.

```
int startPos = s->IndexOf(" ") + 1;
word2 = s->Substring(startPos, s->IndexOf(" ", startPos) - startPos);
```

У цьому фрагменті:

`s->IndexOf(" ") + 1` – індекс початку другого слова (наступний символ після пробілу);

`IndexOf(" ", startPos)` – індекс пробілу після другого слова;

`s->IndexOf(" ", startPos) - startPos` – довжина другого слова.

3) **Insert** – вставляння підрядка в задану позицію поточного рядка.

Наприклад, після виконання команд

```
String ^s = ".NET Framework 4.5";
s = s->Insert(0, "C++");
```

рядок `s` набуде вигляду: `"C++.NET Framework 4.5"`, тобто на початок рядка буде вставлено фрагмент `"C++"`.

4) **Remove** – видалення з рядка певної кількості символів із заданої позиції поточного рядка:

```
Remove (int startIndex, int count)
```

Наприклад, наступний набір команд видалить з рядка друге слово – набір символів між першим і другим пробілами, тобто значення по батькові:

```
String ^name = "Микола Петрович Тараненко";
int foundS1 = name->IndexOf(" ");
int foundS2 = name->IndexOf(" ", foundS1 + 1);
name = name->Remove(foundS1 + 1, foundS2 - foundS1);
```

Якщо другого слова в рядку не буде, видаляться всі символи до кінця рядка.

5) **Replace** – заміна частини рядка. Метод замінює всі входження в рядок певної сукупності символів `oldValue` на іншу сукупність символів `newValue`:

```
Replace (String^ oldValue, String^ newValue)
```

Наприклад, змінити слово `математика` в рядку `s` (якщо воно там є) на нову назву можна командою:

```
s = s->Replace("математика", "вища математика");
```

6) **Методи ToLower і ToUpper** перетворюють усі символи рядка до нижнього чи то до верхнього регістру, при чому методи коректно опрацьовують і латиницю, і кирилицю. Наприклад:

```
String ^s = Console::ReadLine();
Console::WriteLine(s->ToUpper());
```

7) **Методи TrimStart, TrimEnd і Trim** дозволяють позбавитись або початкових, або кінцевих, або і тих і тих пробілів у рядку.

8) **Split** – поділ рядка на порції, залежно від вказаного розділювача. Метод відразу формує масив рядків (`array<String^>`) з виокремленими порціями символів – підрядками, які були розміщені у вихідному рядку між будь з яких зазначених розділювачів, наприклад:

```
String ^s = "Тараненко 12.06.1998 167";
array<Char> ^Сепаратор = { ' ', '\t' };
array<String^> ^subs =
    s->Split(Сепаратор, StringSplitOptions::RemoveEmptyEntries);
```

У наведеному прикладі другий параметр в методі `Split` вказує на те, що порожні елементи не будуть включатися до масиву формованих підрядків. Результа-

том виконання наведеної команди буде масив `subs` з трьох елементів: `subs[0]="Тараненко"; subs[1]="12.06.1998"; subs[2]="167"`. Розділювачами були вказані пробіл і символ табуляції, причому їхня кількість чи поєднання не мають значення, оскільки вони не включаються до набору елементів формованого масиву.

9) **Compare** – порівняння двох рядків. Метод повертає ціле число, яке показує їх відносне положення в порядку сортування: `-1` (якщо перший рядок менше другого), `0` (якщо рядки однакові) або `1` (якщо перший рядок більше другого). При порівнянні рядків відбувається почергове посимвольне порівняння кодів символів у рядках. Наприклад, в умовному операторі

```
if (String::Compare("2+1","1+2")>0) . . .
```

умова буде істинною, оскільки код символу `'2'` є більшим за код символу `'1'`.

Деякі методи для роботи із символами Char

1) **IsDigit** – перевірка на те, чи є символ десятковою цифрою. Якщо символ є цифрою, то результат – `true`, інакше – `false`.

Звичайно, метод використовують разом з оператором `if` у такий спосіб:

```
if (Char::IsDigit(s[i])) . . .
```

2) **IsLetter** – повертає значення `true`, якщо символ є будь-якою літерою Unicode (кирилиця також), інакше – `false`.

3) **IsLower** – повертає значення `true`, якщо символ є літерою нижнього регістру (малою літерою), інакше – `false`.

4) **IsUpper** – повертає значення `true`, якщо символ є літерою верхнього регістру (великою літерою), інакше – `false`.

5) **IsPunctuation** – повертає значення `true`, якщо символ є одним зі знаків пунктуації: `'.'`, `','`, `':'`, `';'`, `'<`, `'>`, `'['`, `']'` та ін.

6) **IsWhiteSpace** – повертає значення `true`, якщо символ є пробілом, інакше – `false`.

7) **Equals** – повертає значення `true`, якщо символ є однаковим (збігається) з порівнюваним значенням, наприклад:

```
for (int i = 0; i < s->Length; i++)  
    if (s[i].Equals('*')) { s = s->Remove(i, 1); i--; }
```

Ці команди видалять всі зірочки з рядка.

Приклади програм з рядками String

Приклад 7.38. Обчислити кількість пробілів і знаків пунктуації у введеному рядку.

Текст програми:

```
using namespace System;  
int main(array<System::String ^> ^args)  
{ Console::WriteLine("Введіть рядок:");  
  String ^s = Console::ReadLine();  
  int nSpace = 0, nPunct=0;
```

```

for (int i = 0; i < s->Length; i++)
{ if (Char::IsPunctuation(s[i])) nPunct++;
  if (Char::IsWhiteSpace(s[i])) nSpace++;
}
Console::WriteLine("У введеному рядку\n{0}\nпрозміщено {1} пробіл(и/ів)
                  і {2} знак(и/ів) пунктуації ", s, nSpace, nPunct);
Console::ReadLine();
return 0;
}

```

Результат виконання програми:

```

Введіть рядок:
На все добре!
У введеному рядку
На все добре!
розміщено 2 пробіл(и/ів) і 1 знак(и/ів) пунктуації

```

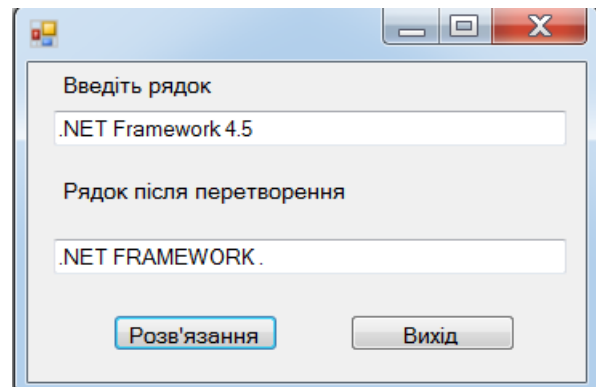
Приклад 7.39. У введеному рядку спочатку видалити всі цифри і перевести символи рядка до верхнього регістру.

Текст програми:

```

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    int kol=0;
    String^ str = textBox1->Text;
    for (int i=str->Length-1; i>=0; i--)
        if (Char::IsDigit(str[i]))
            str = str->Remove(i, 1);
    str = str->ToUpper();
    textBox2->Text = str;
}

```



Приклад 7.40. Замінити в рядку цифри на їхні словесні назви, наприклад: 1 – на "один", 2 – на "два" тощо.

Розв'язок. Для таких заміन доцільно використати метод `Replace()`, але оскільки цей метод може працювати з аргументами – рядками, то спочатку символ слід перетворити до рядка за допомогою методу `ToString()`. Вибір словесної назви тієї чи іншої цифри зручно організувати в операторі `switch`, вказавши в якості ключа сам символ. При цьому, насправді, буде опрацьовуватись код символу.

Текст програми:

```

using namespace System;
int main(array<System::String ^> ^args)
{ String ^s = "2 * 2 = 4 - это всем известно в целом мире,
              \на не 6 и не 5 - это надо знать!";
  Console::WriteLine(s);
}

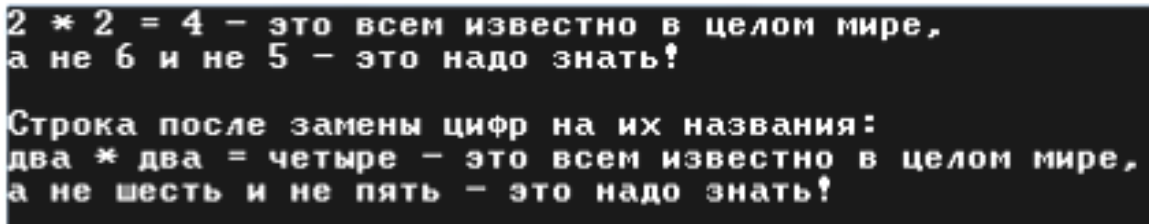
```

```

int i=0; String ^nazva;
for (int i = 0; i < s->Length-1; i++)
    if (Char::IsDigit(s[i]))
    { switch (s[i])
        { case '0': nazva="ноль";break;
          case '1': nazva="один";break;
          case '2': nazva="два";break;
          case '3': nazva="три";break;
          case '4': nazva="четыре";break;
          case '5': nazva="пять";break;
          case '6': nazva="шесть";break;
          case '7': nazva="семь";break;
          case '8': nazva="восемь";break;
          case '9': nazva="девять";break;
        }
        s = s->Replace(s[i].ToString(), nazva);
    }
Console::WriteLine("\nСтрока после замены цифр на их названия:\n"+s);
Console::ReadLine();
return 0;
}

```

Результат виконання програми:



```

2 * 2 = 4 - это всем известно в целом мире,
а не 6 и не 5 - это надо знать!

Строка после замены цифр на их названия:
два * два = четыре - это всем известно в целом мире,
а не шесть и не пять - это надо знать!

```

Приклад 7.41. У введеному рядку вставити після кожного символу '*' його порядковий номер та номер позиції цього символу у рядку.

Текст програми:

```

using namespace System;
int main(array<System::String ^> ^args)
{ Console::WriteLine("Введіть рядок:");
  String ^s = Console::ReadLine();
  int kol = 0, i;
  for (i = 0; i <= s->Length-1; i++)
      if (s[i].Equals ('*'))
      { kol++;
        s = s->Insert(i+1, " (nom="+kol.ToString()+" pos="+i.ToString()+") ");
      }
  Console::WriteLine("\nРезультат:\n"+s);
  Console::ReadLine();
  return 0;
}

```

Результат виконання програми:

```
Введіть рядок:
*qw*33*

Результат:
* (nom=1 pos=0) qw* (nom=2 pos=18) 33* (nom=3 pos=37)
```

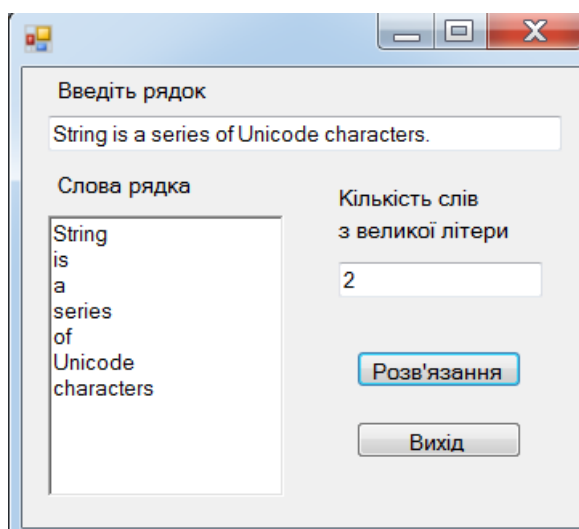
Приклад 7.42. Ввести рядок, вивести окремо кожне зі слів цього рядка та обчислити кількість слів рядка, які починаються з великої літери.

Розв'язок. Зверніть увагу, що у наведеному програмному коді у методі Split параметр StringSplitOptions::RemoveEmptyEntries використано для усунення порожніх рядків, які можуть виникнути у разі введення подвійних пробілів тощо.

Текст програми:

```
private: System::Void button1_Click(System::Object^sender, System::EventArgs^e)
{
    int kol = 0;
    String ^str = textBox1->Text;
    String ^delimStr = " ,.:\\t";
    array<Char>^ delimiter = delimStr->ToCharArray( );
    array<String>^ ^words;
    words = str->Split(delimiter, StringSplitOptions::RemoveEmptyEntries);
    for (int i=0; i<words->Length; i++)
    {
        richTextBox1->Text += words[i]+"\\r\\n";
        if (Char::IsUpper(words[i][0])) kol++;
    }
    textBox2->Text = Convert::ToString(kol);
}
```

Результат виконання програми:



Приклад 7.43. У введеному рядку перевести голосні латинські літери до верхнього регістру.

Текст програми:

```
using namespace System;

int main(array<System::String ^> ^args)
{ Console::WriteLine("Введіть рядок:");
  String ^s = Console::ReadLine();
  for (int i = 0; i < s->Length; i++)
    if (s[i]=='a' || s[i]=='o' || s[i]=='e' || s[i]=='i' || s[i]=='u')
    { Char c = Char::ToUpper(s[i]);
      s = s->Remove(i,1);
      s = s->Insert(i,c.ToString());
    }
  Console::WriteLine("\nРезультат:\n" + s);
  Console::ReadLine();
  return 0;
}
```

```
Введіть рядок:
Hello, world!

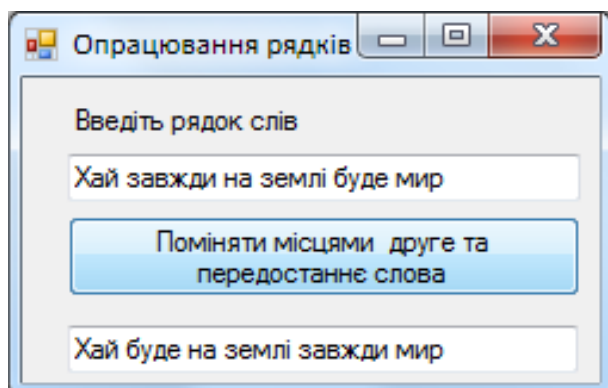
Результат:
HEllo, wOrld!
```

Приклад 7.44. У введеному рядку поміняти друге та передостаннє слова.

Текст програми:

```
private: System::Void button1_Click(System::Object^sender, System::EventArgs^e)
{ String^ s = textBox1->Text->Trim();
  String ^dS = " .,:!?\t";
  array<Char>^ del = dS->ToCharArray();
  array<String>^ ^words = s->Split(del,
StringSplitOptions::RemoveEmptyEntries);
  int n = s->IndexOf(words[words->Length-2]); // позиція передостаннього слова
  s = s->Remove(n, words[words->Length-2]->Length); // видалити передостаннє слово
  s = s->Insert(n, words[1]); // поставити замість видаленого друге слово
  n = s->IndexOf(words[1]); // позиція другого слова
  s = s->Remove(n, words[1]->Length); // видалити друге слово
  s = s->Insert(n, words[words->Length-2]); // поставити туди передостаннє слово
  textBox2->Text = s;
}
```

Результат виконання програми:

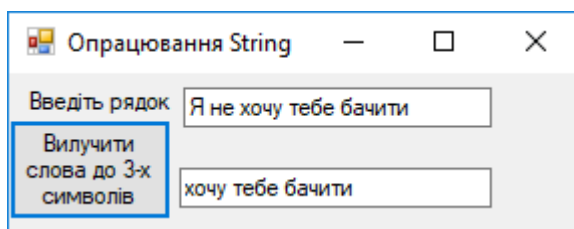


Приклад 7.45. Ввести рядок і вилучити з нього всі слова, довжина яких є менше за три символи.

Текст програми:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{ String^ S = textBox1->Text;
  S=S+" "; // Приєднання пробілу, щоб «не втратити» останнє слово рядка
  int p=0, i, dl; // p – перша літера слова
  // Оскільки довжина рядка буде змінюватися при вилучанні слів, то не можна
  // в умові виходу із циклу ставити її заздалегідь обчислене значення
  for (i=0; i<S->Length; i++)
  { if (S[i]==' ') // Якщо символ є пробілом, це є ознакою завершення слова
    { dl=i-p; // Обчислення довжини слова
      if (dl<3) // Слова з довжиною меншою за 3
      { S=S->Remove(p, i-p+1); // вилучаються,
        i=p-1; // а позиція переміщується на пробіл перед вилученим словом,
      } // інакше (якщо слово не вилучено),
      else p=i; // p стає номером першої літери наступного слова
    }
  }
  textBox2->Text=S; // Виведення рядка після вилучення слів
}
```

Результат виконання програми:

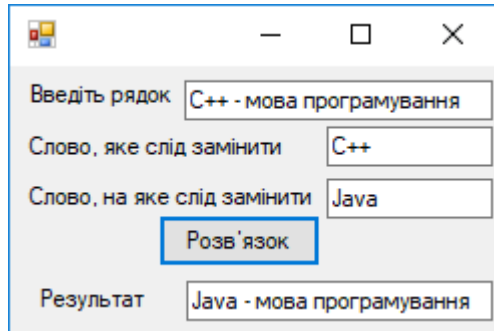


Приклад 7.46. Ввести рядок та замінити у рядку всі входження одного введеного слова на інше.

Текст програми:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
  String ^S, ^s1, ^s2;
  S=textBox1->Text+" ";
  s1=textBox2->Text; s2=textBox3->Text;
  while (S->IndexOf(s1)>=0) // Допоки слово, яке треба замінити, є в рядку,
  { int k=S->IndexOf(s1); // визначається позиція цього слова,
    S=S->Remove(k, s1->Length); // воно вилучається і на його місце
    S=S->Insert(k, s2); // вставляється нове слово
  }
  textBox4->Text=S; // Виведення нового рядка
}
```

Результат виконання програми:



Приклад 7.47. Ввести рядок і відсортувати слова у рядку: а) за абеткою, б) за спаданням довжин слів.

Розв'язок. Оскільки завдання складається з двох явних частин, а також для більшої зрозумілості програмного коду організуємо розв'язання в окремих функціях.

Текст програми:

```
// Функція сортування масиву слів за абеткою
void sort_alf( array<String^> ^arr, int kol)
{ bool f; int i;
  do
  { f=true;
    for (i=0; i<kol-1; i++)
      if (String::Compare(arr[i],arr[i+1])>0)
      { String^ temp=arr[i];
        arr[i]=arr[i+1];
        arr[i+1]=temp;
        f=false;
      }
    } while (f==false);
}

// Функція сортування масиву слів за спаданням довжин
void sort_len( array<String^> ^arr, int kol)
{ bool f; int i; String^ temp;
  do
  { f=true;
    for (i=0; i<kol-1; i++)
      if (arr[i]->Length < arr[i+1]->Length)
      { temp=arr[i]; arr[i]=arr[i+1]; arr[i+1]=temp;
        f=false;
      }
    } while (f==false);
}

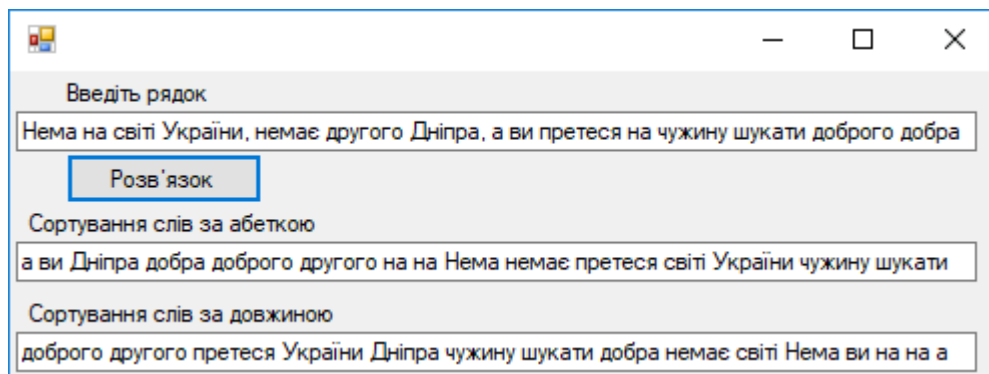
// Функція копіювання масиву слів у рядок
String^ copy_array(array<String^> ^arr, int kol)
{ int i; String^ S="";
  for (i=0; i<kol; i++)
```

```

    { S=S+arr[i];
      if( i!=kol-1) S=S+" ";
    }
    return S;
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    String ^S=textBox1->Text+" ", ^rez;
    int i, j=0, kol=0, N=S->Length;
    for (i=0; i<N; i++)
        if (Char::IsWhiteSpace(S[i])==false && Char::IsWhiteSpace(S[i+1]))
            kol++;
    array<Char> ^Сепаратор = { ' ', '\t', ',', '.', '!', '?' };
    array<String^> ^arr= S->Split(Сепаратор,
                                StringSplitOptions::RemoveEmptyEntries);
    sort_alf(arr, kol); // Виклик функції сортування масиву за абеткою
    rez=copy_array(arr,kol); // Виклик функції копіювання масиву у рядок
    textBox2->Text=rez; // Виведення рядка після сортування за абеткою
    sort_len(arr,kol); // Виклик функції сортування слів за зростанням довжин
    rez=copy_array(arr,kol);
    textBox3->Text=rez; // Виведення рядка після сортування
    delete []arr;
}

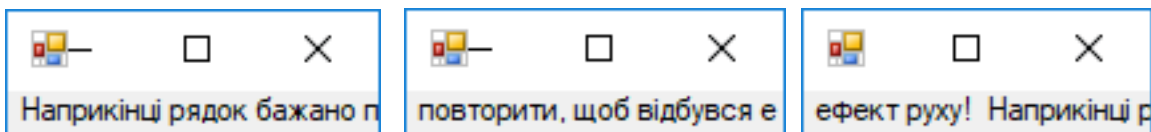
```

Результат виконання програми:



Приклад 7.48. Створити рядок, символи якого пересуваються, «біжать» екраном.

Розв'язок. Розмістимо на формі лише два елементи керування: `label1`, який показуватиме на формі певну змінювану частину рядка `GoString`, та `Timer1`, який задаватиме частоту змінювань символів. Ніякого попереднього встановлення значень властивостей цих елементів наведена програма не потребує.



Текст програми:

```
static int P=0;
// Функція виведення частини рядка викликається кожні 1/5 секунди
private: System::Void timer1_Tick(System::Object^ sender, System::EventArgs^e)
{
    timer1->Interval=200;    // Частота змінювань символів – 1/5 секунди
    // Рядок, який рухатиметься
    String^ GoString=" Наприкінці рядок бажано повторити, щоб
                        відбувся ефект руху! Наприкінці рядок бажано";
    int LengthGoString=35;    // Ширина вікна виведення
    // Виведення частини рядка, розпочинаючи з позиції P.
    label1->Text=GoString->Substring(P, LengthGoString);
    P++;    // Перехід до наступного символу
    // Якщо решта рядка є коротша за ширину вікна виведення, перейти
    if (GoString->Length - LengthGoString < P) P=1; // на початок рядка
}
```

7.3 Розширені символьні типи

7.3.1 Тип `wchar_t`

Алфавіт більшості європейських мов може бути подано одnobайтовими числами (тобто кодами в діапазоні від 0 по 255). Для більшості кодувань перші 127 кодів є однаковими і становлять набір ASCII-кодів: арабські цифри, великі й малі літери латиниці, спеціальні символи й розділові знаки (див. додаток А). Друга половина кодів – від 128 по 255 – відводиться під літери тієї чи іншої мови. Фактично друга половина кодової таблиці інтерпретується по-різному в різних кодуваннях для різних країн.

Проте для таких мов, як китайська, японська і деякі інші, одного байта вже недостатньо, щоб закодувати всі їхні символи, оскільки алфавіти цих мов нараховують понад 127 і навіть понад 255 символів. Тому для кодування символів стали відводити вже два байти, а таке інтернаціональне кодування назвали **Unicode**. Перші 127 символів Unicode збігаються з ASCII-символами.

У мові C++ для опрацювання двобайтових символів існує спеціальний тип `wchar_t`. Цей тип іноді називають типом «широких» символів, і деталі його реалізації можуть варіюватися від компілятора до компілятора, зокрема може змінюватися і кількість байтів, яка відводиться під один символ; зазвичай вона відповідає типові `short` (2 байти). Константи типу `wchar_t` записуються з префіксом `L` у вигляді `L'ab'`:

```
wchar_t c=L'ab';
```

Рядкові константи типу `wchar_t*` записуються також з префіксом `L`, наприклад: `L"Gates"`. Існує ціла низка спеціальних функцій для роботи з цими рядками. Майже всі функції для рядків `char*` мають аналоги для `wchar_t*`. Наприклад, функція `wcslenn()` визначає довжину рядка типу `wchar_t`, команди `wcin`

та `wcout` вводять та виводять ці рядки, функції `swscanf()` та `swprintf()` вводять і виводять їх у заданому форматі, функція `wscat()` сполучає рядки, функції `wchr()`, `wspbrk()` і `wsstr()` здійснюють пошук символів в таких рядках тощо.

Приклад 7.49. Демонструється опрацювання `wchar_t` рядків.

Текст програми:

```
#include <iostream>
#include <string>
#include <wchar.h>
using namespace std;
int main()
{ setlocale(0, ".1251");
  wchar_t s1[100]=L"Hello "; // Оголошення із ініціалізацією рядків
  wchar_t *s2=L"world";
  int n=wcslen(s1);           // Визначення довжини n рядка s1
  wcout << n << " : " << s1 << endl; // Виведення рядка s1
  n=wcslen(s2);
  wcout << n << " : " << s2 << endl;
  wscat(s1, s2);             // Долучення s2 до s1
  n=wcslen(s1);
  wcout << n << " : " << s1 << endl;
  cin.get();
  return 0;
}
```

Результати виконання програми:

```
6 : Hello
5 : world
11 : Hello world
```

7.3.2 Тип `wstring`

Тип `wstring` є еквівалентом масиву «широких» символів кодування Unicode типу `wchar_t` – `vector<wchar_t>`.

Для введення-виведення рядків `wstring` слід використовувати потокові оператори `wcin` і `wcout` відповідно, замість `cin` і `cout`, що призначені для `string`.

Посимвольне опрацювання рядків `wstring` здійснюється подібно до інших типів рядків. Можливим є перетворення рядка до `wstring` типу `wchar_t*`:

```
wstring ws=L"На все добре!";
wchar_t *ws = (wchar_t*)ws.c_str();
```

Як приклад розглянемо програмний код для оголошення двох рядків, введення одного з них і формування третього рядка в результаті зчеплення двох рядків, при чому перша літера цього рядка буде великою, незалежно від того чи була вона літерою латиниці, чи то кирилиці:

```
wstring s, s1, s2=L"на все добре!";
setlocale(0, ".1251"); wcout << L"Введіть рядок : ";
setlocale(0, ".ОСР");  getline(wcin,s1);
```

```
setlocale(0, ".1251");  
if ((s1[0]>=L'a' && s1[0]<=L'z') || (s1[0]>=L'а' && s1[0]<=L'я'))  
    s1[0]-=32;  
s=s1+L" "+s2;  
wcout << s <<endl;
```

Результатом цих команд можуть бути такі рядки:

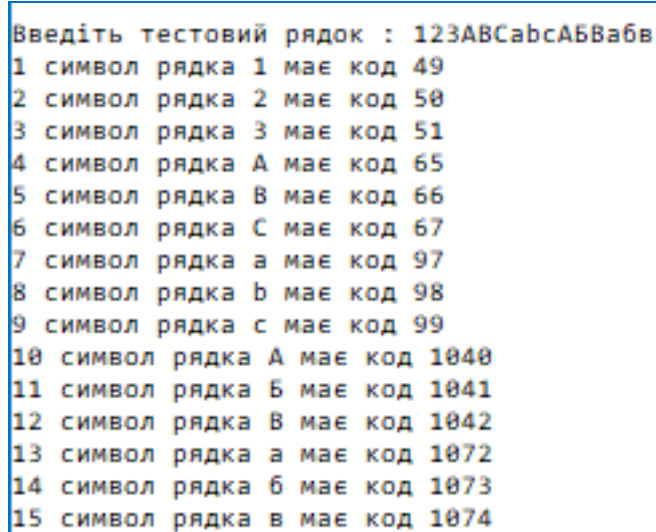
```
Введіть рядок : моя люба,  
Моя люба, на все добре!
```

Приклад 7.50. Ввести рядок і вивести значення кодів (Unicode) кожного символу введенного wstring-рядка:

Текст програми:

```
#include <iostream>  
#include <string>  
#include <wchar.h>  
using namespace std;  
int main(int argc, char *argv[])  
{ system("color F0");  
  setlocale(0, ".1251");  wcout << L"Введіть тестовий рядок : ";  
  wstring ws(L"\n");  
  setlocale(0, ".OCP");  getline(wcin, ws);  
  setlocale(0, ".1251");  
  for (int i=0; i<wcslen(ws.c_str()); i++)  
      wcout<<i+1<<L" символ рядка "  
        << ws[i] << L" має код "  
        << (int)ws[i] <<endl;  
  cin.get();  
  return 0;  
}
```

Результат виконання програми:



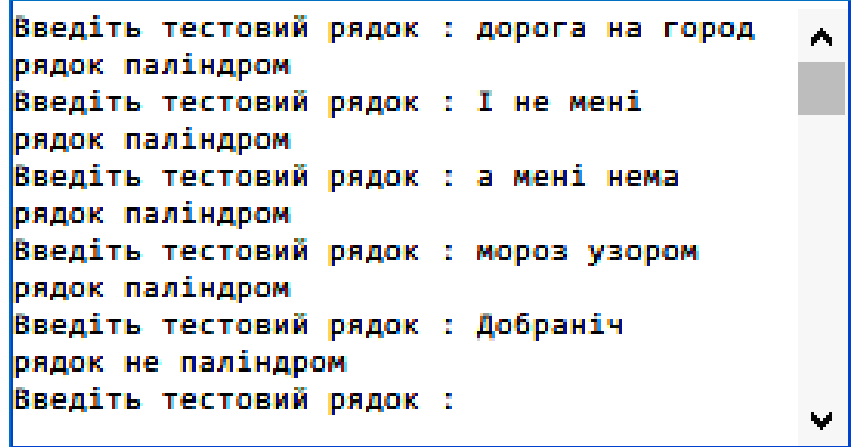
```
Введіть тестовий рядок : 123ABСabсАБВабв  
1 символ рядка 1 має код 49  
2 символ рядка 2 має код 50  
3 символ рядка 3 має код 51  
4 символ рядка A має код 65  
5 символ рядка B має код 66  
6 символ рядка C має код 67  
7 символ рядка a має код 97  
8 символ рядка b має код 98  
9 символ рядка c має код 99  
10 символ рядка A має код 1040  
11 символ рядка Б має код 1041  
12 символ рядка В має код 1042  
13 символ рядка а має код 1072  
14 символ рядка б має код 1073  
15 символ рядка в має код 1074
```


Приклад 7.51. Ввести рядок і перевірити, чи є він паліндромом без урахування пробілів і знаків пунктуації.

Текст програми:

```
#include <iostream>
#include <string>
#include <wchar.h>
using namespace std;
int main(int argc, char *argv[])
{
    system("color F0");
    setlocale(0, ".1251");
    wstring w(L"\n");
    while (w.size() > 0)
    {
        wcout << L"Введіть тестовий рядок : ";
        getline(wcin, w);
        if (w.size() == 0) break;
        bool poli;                                // ознака паліндрома
        wchar_t *s = (wchar_t*)w.c_str(); // початок рядка
        wchar_t *pb = s;
        wchar_t *pe = pb + wcslen(s) - 1; // кінець рядка
        do
        {
            while (*pb == ' ') pb++;
            while (*pe == ' ') pe--;
            // перетворення заголовних літер на малі
            poli = towlower(*pb) == towlower(*pe);
        } while (poli && ++pb <= --pe);
        wcout << L"рядок " << (poli ? L"" : L"не ") << L"паліндром" << endl;
    }
    cin.get();
    return 0;
}
```

Результат виконання програми:



```
Введіть тестовий рядок : дорога на город
рядок паліндром
Введіть тестовий рядок : I не мені
рядок паліндром
Введіть тестовий рядок : а мені нема
рядок паліндром
Введіть тестовий рядок : мороз узором
рядок паліндром
Введіть тестовий рядок : Добраніч
рядок не паліндром
Введіть тестовий рядок :
```

Питання та завдання для самоконтролю

- 1) Дати означення терміну ASCII.
- 2) Яка є кількість ASCII-кодів? І чому саме?
- 3) Скільки байтів виділяється для змінної типу `char`?
- 4) Які операції можна виконувати над символами?
- 5) Назвсти вирази, які матимуть значення 1 (`true` – істина – «так»):
 - a) `'0' < 'y'`
 - б) `'5' > 'f'`
 - в) `'F' > 'f'`
 - г) `'w' > 'W'`
 - д) `' ' > '0'`
 - е) `'я' > 'ю'`
- 6) Як у програмі змінити регістр символів латиниці?
- 7) Навести відомі способи оголошення C-рядка.
- 8) В який спосіб можна звернутися до окремого символу рядка?
- 9) Як порівняти два C-рядки? Навести приклад.
- 10) Чи можна присвоїти C-рядку частину іншого? Якщо так, навести приклад.
- 11) Чи можна з'єднувати рядки, якщо так, то в який спосіб?
- 12) Навести способи введення C-рядка у консолі.
- 13) Пояснити, що виведеться на екран після виконання фрагмента програми:

```
char s1[]="слово", *s2=new char[5];
strncpy(s2, s1, 3);
puts(s2);
```
- 14) Як виправити логічну помилку, припущену у завданні 13?
- 15) Навести і пояснити фрагмент коду для видалення із C-рядка трьох символів, розпочинаючи з п'ятої позиції.
- 16) У рядку символів визначити кількість повторювань кожного слова та видалити дублікати слів. Слова відокремлюються пробілами.
- 17) У рядку символів довільного розміру видалити усі слова, довжина яких перевищує N літер.
- 18) Скопіювати до нового рядка частину рядка від початку до:
 - a) першого пробілу;
 - б) другого пробілу;
 - в) останнього пробілу.
- 19) Які типи рядків підтримуються у Visual C++?
- 20) Пояснити відмінності символів `char` від `Char` і рядків `char*` від `String`.
- 21) Скільки байтів виділяється для змінної класу `String`?
- 22) Скільки байтів виділяється для одного символу `String`-рядка?
- 23) Навести фрагменти коду для заміни всіх малих літер 'a' на коми:
 - a) у C-рядку;
 - б) у рядку `string`;
 - в) у рядку `String`.
- 24) Навести приклад оголошення та введення рядка класу `String`.
- 25) Яка функція Visual C++ перетворює ціле число на рядок `String`? А яка навпаки? Навести приклади.
- 26) Що виведеться на екран після виконання команд:

```
String ^s = ".NET Framework 4.5";
Console.WriteLine(s->ToUpper());
```

Скільки символів рядка `s` після цього залишиться незмінними?

27) Що виведеться на екран після виконання команд:

```
String ^s = ".NET Framework 4.5";  
s = s->Insert(0, "C++");  
Console::WriteLine(s);
```

28) Що виведеться на екран після виконання команд:

```
String ^s = ".NET Framework 4.5";  
s = s->Remove(5, 9);  
Console::WriteLine(s);
```

29) Записати команду, яка дозволить знайти у рядку `s` (див. завдання 28) індекс першого пробілу?

30) Записати команду, яка дозволить вставити на початок рядка `s` слово "good"?

РОЗДІЛ 8

ФУНКЦІЇ

8.1 Правила організації функцій

Коли програма стає зовеликою за обсягом і складною для сприйняття, є сенс розділити її за змістом на невеликі логічні частини, називані *функціями*, кожна з яких виконуватиме певне завдання. Унаслідок цього програма стане більш легкою і для розуміння при створюванні, і для процесу налагодження. Окрім того, створення функції позбавляє потреби створювання двох, а то й і більшої кількості, майже однакових фрагментів програмного коду для розв'язання однакових завдань за різних вхідних даних. Розподіл програми на функції є базовим принципом структурного програмування.

Функція – це незалежна іменована частина програми, яка може багаторазово викликатися з інших частин програми, маніпулювати даними та повертати результати. Кожна функція має власне ім'я, за яким здійснюють її виклик. Розрізняють два основні різновиди функцій: 1) стандартні убудовані функції, які є складовою частиною мови програмування, наприклад: `sin()`, `pow()` тощо і 2) функції, які створюються безпосередньо користувачем для власних потреб.

Створювана у програмі функція повинна бути *оголошеною* і *визначеною*. Оголошення функції має бути написаним у програмі раніш за її використання. Визначення може перебувати у будь-якому місці програми, за винятком тіла (середини) інших функцій.

Оголошення функції складається з *прототипу* (чи *заголовка*) і має форму:

```
[клас] <тип результату> <ім'я функції>
      (<перелік формальних аргументів із зазначенням типу>);
```

Наприклад, оголошення функції обчислення середньоарифметичного двох цілих чисел може мати вигляд:

```
double seredne (int a, int b);
```

де `seredne` – ім'я функції;

`a` і `b` – формальні вхідні аргументи (параметри), які за виклику набувають значень фактичних параметрів;

`double` – тип функції, який безпосередньо є типом результату виконання операторів функції. Результат повертається оператором `return`.

Прототип функції може бути розташовано як безпосередньо у програмі, так і в заголовному файлі (див. далі п. 9.2).

Окрім оголошення, кожна функція повинна мати визначення (реалізацію).

Визначення функції, окрім заголовку, містить тіло функції (команди, які виконує функція) і команду виходу із функції з можливим повертанням результату в точку виклику `return`:

```

<тип функції> <ім'я функції> (<перелік аргументів>)
{ <тіло функції>
    return <результат обчислень>;
}

```

Наприклад, визначення функції `seredne` може бути таким:

```

double seredne (int a, int b)
{ double sr;
  sr=(a+b)/2.0;
  return sr;
}

```

Наведена функція обчислює `sr` – середньоарифметичне двох цілих чисел `a` та `b` і повертає його значення у точку виклику.

У разі, коли визначення функції розміщено у програмі раніш за точку виклику, писати окремо оголошення і окремо реалізацію цієї функції немає потреби і оголошення функції можна уникнути.

Якщо функції виконують певні обчислення й дії, які не потребують повертання результатів, за їхній тип вказують тип `void` (тобто порожній, без типу). У таких функціях оператор `return` може бути відсутнім чи записуватись без значення, яке повертається. Яскравим прикладом таких функцій є відгуки подій у Visual C++, наприклад:

```

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
}

```

У поданому нижче прикладі створювання власної консольної функції `max3()` всі обчислення для пошуку максимального з трьох дійсних чисел `x`, `y` та `z`, а також дії стосовно виведення результатів обчислень відбуваються у функції і немає потреби повертати їх через `return`:

```

void max3(double x, double y, double z)
{ if (x>y && x>z) cout<<x;
  else
    if (y>x && y>z) cout<<y;
    else cout<<z;
}

```

Приклади створювання функцій з типом `void` вже розглядались у розд. 5 при опрацюванні числових масивів у функціях.

Виконання функції розпочинається, коли у тексті програми зустрічається оператор виклику, тобто ім'я функції з фактичними параметрами у дужках. При цьому формальні параметри послідовно набувають значень фактичних: значення першого фактичного аргументу копіюється у перший формальний аргумент, другий – у другий тощо. А тому дуже важливими чинниками є кількість, порядок записування і відповідність типів всіх аргументів, інакше це може призвести до серії помилок.

Якщо функцію оголошено з будь-яким типом, окрім типу `void`, тобто вона повертає певний результат, при викликанні такої функції її варто присвоїти змінній того ж самого типу, що й сама функція, чи то вивести на екран за допо-

могою оператора виведення. Наприклад, функцію `seredne()` може бути викликано в один зі способів:

```
int a=5, x=2, y=-3; double z, s1, s2;  
1) z = seredne(5, 4);           // z = 4.5  
2) s1 = seredne(a, 11);         // s1 = 8  
3) s2 = seredne(x, y);          // s2 = -0.5
```

У першому наведеному виклику числа 5 та 4 є фактичними параметрами, які підставляються у дужки замість формальних параметрів `a` та `b`:

```
double seredne(int a, int b)  
    z = seredne( 5,    4);
```

У другому виклику першим фактичним параметром є змінна `a`, а другим – число 11. У третьому – обидва фактичних параметри є змінними.

Вищенаведена функція `max3()` має тип `void`, тому її виклик може мати вигляд

```
max3(5, 1, 14);
```

У результаті цієї команди здійсниться виклик функції `max3()`, яка з трьох чисел: 5, 1, 14 віднайде максимальне (14) і його буде виведено на екран.

Функція може бути оголошена *убудованою* (`inline`):

```
inline int modul(int x)  
{  
    if (x<0) return -x;  
    else return x;  
}
```

У такому разі компілятор підставляє у точку виклику тіло цієї функції.

Основні правила організації функцій:

1) Фактичними параметрами можуть бути константи, змінні чи вирази. В останньому разі спочатку обчислюється значення виразу, а потім результат передається у функцію.

2) Імена змінних, які є фактичними параметрами, можуть не збігатися з іменами формальних параметрів.

3) Типи змінних, які є фактичними параметрами, мають збігатися з типами відповідних формальних параметрів чи то мати можливість бути перетвореними до типів формальних параметрів.

4) Якщо кількість чи типи фактичних параметрів не збігаються з формальними параметрами, це призведе до помилки з відповідним повідомленням.

5) Послідовність змінних, які є фактичними параметрами, має збігатися з послідовністю формальних параметрів.

6) У прототипі функції необов'язково задавати імена формальних параметрів, достатньо їх оголосити, наприклад у такий спосіб:

```
double seredne(int, int);
```

7) Якщо функція не отримує жодного параметра, слід у заголовку функції ставити порожні дужки (опустити дужки не можна) чи записати у дужках `void`:

```
double func();  
double func(void);
```

8) Тип значення, яке повертає функція, може бути яким-завгодно, але не може бути масивом чи функцією. Наприклад, таке оголошення є помилкове:

```
int [10] func(); // Помилка!
```

Якщо функція має повернути масив, то можна оголосити її тип як вказівник на перший елемент масиву (кількість елементів цього масиву має бути відомою):

```
int * func();
```

9) C++ дозволяє існування у програмі функцій з однаковим ім'ям, але різними параметрами.

Функції використовують пам'ять зі стека програми. Певна область стека надається функції й залишається пов'язаною з нею до завершення її роботи. Після завершення роботи надана для функції пам'ять звільнюється й може бути зайнята іншою функцією.

Усі величини, оголошені всередині функції, а також її параметри, є *локальними*. Областю їхньої дії є функція, тобто для решти функцій програми ці змінні та їхні значення залишаються невідомими. Глобальні змінні є відомими всім функціям програми, кожна з функцій може змінювати значення таких змінних. Використання глобальних змінних часто призводить до помилок, які важко відшукати. Тому слід уникати використання глобальних змінних, якщо це не є нагальною потребою.

При викликанні функції, як і при вході до кожного блока, у стеку виділяється пам'ять під локальні автоматичні змінні. При виході з функції пам'ять, яку було виділено під локальні змінні, звільнюється. За спільної роботи функції обмінюються даними. Це можна здійснити за допомогою *глобальних* змінних, через *параметри* й через *значення*, які повертає функція.

У розглянутому нижче прикладі змінні з ім'ям *x* визначені одразу у двох функціях – в *main()* і в *Sum()*, що не заважає компіляторові розрізняти їх поміж собою:

```
int Sum(int A, int B)
{ // Локальну змінну x і параметри A та B видно лише у тілі функції Sum()
  int x = A + B;
  return x;
}

void main()
{ int x = 2, y = 4; // x, y - локальні змінні для main()
  cout << Sum(x, y);
}
```

Глобальні змінні є видимими у всіх функціях, де не описані локальні змінні з тими ж іменами, тому використовувати їх для передавання даних між функціями дуже легко. Якщо імена глобальної й локальної змінної збігаються, перевага надається локальній змінній. Проте використовувати змінні з однаковими іменами не рекомендовано, оскільки це ускладнює налагодження програми. Слід прагнути до того, щоб функції були максимально незалежними, а їхній інтерфейс повністю визначався би прототипом функції.

Приклади програм з функціями

Приклад 8.1. Обчислити значення виразу $x = \frac{\sqrt{6}+6}{2} + \frac{\sqrt{13}+13}{2} + \frac{\sqrt{21}+21}{2}$.

Розв'язок. Усі три доданки заданої формули є схожі один з одним. Тому кожен з них можна записати за допомогою спільної формули: $\frac{\sqrt{a}+a}{2}$, де a – число, яке у першому доданку дорівнює 6, у другому – 13, а у третьому – 21. Служно є створити функцію, параметром якої буде дійсне число a і яка обчислюватиме значення цієї формули. У програмі функцію буде викликано тричі для кожного з доданків: 6, 13, 21.

Текст функції та її виклику в основній програмі:

```
#include <iostream>
using namespace std;
double f(double a)
{
    return (sqrt(a)+a)/2;
}
int main(int argc, char *argv[])
{
    double x = f(6) + f(13) + f(21);
    cout<<"x = "<< x<<endl;
    system("pause>>void");
    return 0;
}
```

Результат виконання програми:

x = 25.3188

Приклад 8.2. Обчислити значення виразу $x = \frac{13+\sqrt{7}}{7+\sqrt{13}} + \frac{15+\sqrt{12}}{12+\sqrt{15}} + \frac{21+\sqrt{32}}{32+\sqrt{21}}$.

Розв'язок. Формула цього прикладу схожа на формулу попереднього завдання і відрізняється лише тим, що функція тут матиме два параметри.

Текст функції та основної програми:

```
#include <iostream>
using namespace std;
double f(double a, double b)
{ return (a+sqrt(b))/(b+sqrt(a));
}
int main(int argc, char *argv[])
{ double x = f(13,7) + f(15,12) + f(21,32);
  cout<<"x = "<< x<<endl;
  system("pause>>void");
  return 0;
}
```

Результат виконання програми:

x = 3.36716

Приклад 8.3. Обчислити значення виразу $z = \frac{2 \cdot 5! + 3 \cdot 8!}{6! + 4!}$.

Розв'язок. У формулі чотири рази зустрічається факторіал. Тому доречно обчислити значення факторіалу у функції і викликати його відповідно чотири рази для різних параметрів.

Текст функції та основної програми:

```
#include <iostream>
using namespace std;

long fact(int n)
{ long c=1;
  for(int i=1; i<=n; i++) c*=i;
  return c;
}
int main(int argc, char *argv[])
{
  system("color F0");
  double z=(2.0*fact(5)+3*fact(8))/(fact(6)+fact(4));
  cout<<"z = "<< z << endl;
  system("pause>>void");
  return 0;
}
```

Результат виконання програми:

x = 162.9032258

Приклад 8.4. Віднайти периметр трикутника, заданого координатами вершин.

Розв'язок. Для обчислення довжини сторони трикутника доцільно створити функцію і викликати її тричі для кожної зі сторін. Ця функція обчислюватиме довжини відрізка, заданого координатами точок, які він з'єднує, за формулою

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

Функція `segment()` має чотири параметри – координати обох кінців відрізка. Результат функції – дійсне число (довжина відрізка).

Текст функції та основної програми:

```
#include <iostream>
using namespace std;
double segment(double x1, double y1, double x2, double y2)
{ return sqrt(pow(x2-x1,2)+pow(y2-y1,2)); }
```

```

int main()
{ double x1, y1, x2, y2, x3, y3, P;
  setlocale(0, ".1251");
  cout<<"Перша вершина:\n";
  cout<<"Введіть координату x: "; cin>>x1;
  cout<<"Введіть координату y: "; cin>>y1;
  cout<<"Друга вершина:\n";
  cout<<"Введіть координату x: "; cin>>x2;
  cout<<"Введіть координату y: "; cin>>y2;
  cout<<"Третя вершина:\n";
  cout<<"Введіть координату x: "; cin>>x3;
  cout<<"Введіть координату y: "; cin>>y3;
  P=segment(x1,y1,x2,y2)+segment(x2,y2,x3,y3)+ segment(x1,y1,x3,y3);
  cout<<"Периметр дорівнює: "<<P<<endl;
  system("pause>>void");
  return 0;
}

```

Результати виконання програми:

```

Перша вершина:
Введіть координату x: 3
Введіть координату y: 5
Друга вершина:
Введіть координату x: 1
Введіть координату y: 7
Третя вершина:
Введіть координату x: 4
Введіть координату y: 2
Периметр дорівнює: 11.8217

```

8.2 Способи передавання параметрів до функцій

Значення автоматичних локальних змінних між викликами однієї й тієї самої функції не зберігаються. Якщо значення певних локальних змінних треба зберігати, при їхньому оголошенні слід використовувати специфікатор **static**:

```

#include <iostream>
using namespace std;
void f(int a)
{ int m=0;
  cout<<"n m p\n";
  while (a--)
  { static int n=0;
    int p=0;
    cout<<n++<<" " <<m++<<" " <<p++<<"\n";
  }
}

```

```
int main()
{ f(3);
  f(2);
  system("pause>>void");
  return 0;
}
```

Статична змінна *n* оголошується й ініціалізується *одноразово* за першого виконання оператора, який містить її визначення. Автоматична змінна *m* оголошується й ініціалізується за *кожного* входження до функції. Автоматична змінна *p* оголошується й ініціалізується за *кожного* входження до блока циклу. Програма виведе на екран:

```
n m p
0 0 0
1 1 0
2 2 0

n m p
3 0 0
4 1 0
```

При викликанні функції першою чергою обчислюються вирази, які стоять на місці фактичних параметрів; потім у стеку виділяється пам'ять під формальні параметри функції відповідно до їхнього типу і кожному з них присвоюється значення відповідного фактичного параметра. При цьому перевіряється відповідність типів і, за потреби, виконуються їхні перетворювання. За невідповідності типів видається діагностичне повідомлення.

Існує два способи передавання параметрів до функції: за значенням і за адресою.

За передавання *за значенням* до стека заносяться *копії* фактичних параметрів і оператори функції працюють з цими копіями. Доступу до вихідних значень параметрів у функції немає, а, отже немає й можливості їх змінити.

За передавання *за адресою* до стека заносяться *копії адрес* параметрів, а функція здійснює доступ до комірок пам'яті за цими адресами і може змінювати вихідні значення параметрів. Передавати параметри за адресою можна у два способи: за допомогою *вказівника* і *посилання*.

За обох способів до функції передається адреса параметра, який зазначено при викликанні. При передаванні вказівника всередині функції виникає потреба використовувати явно операції розіменування чи то отримання адреси, що робить менш зрозумілим текст коду функції і більш складним налагодження функції. За передавання за посиланням до функції передається посилання, тобто синонім імені, внаслідок чого всередині функції всі звертання до параметра неявно розіменовуються.

Отже, використання посилань замість вказівників зоптимізовує читабельність програми, позбавляючи потреби застосовувати операції отримання адреси й розіменування. Використання посилань замість передавання за значенням є більш ефективне, оскільки не потребує копіювання параметрів, а це є надто важливо при передаванні структур даних великого обсягу.

Розглянемо приклад функції з передаванням параметрів за посиланням:

```
void duplicate(int& a, int& b, int& c)
{ a*=2; b*=2; c*=2;
}
int main ()
{ int x=1, y=3, z=7;
  duplicate(x, y, z);
  cout << "x=" << x << ", y=" << y << ", z=" << z;
  system("pause>>void");
  return 0;
}
```

Результат:

x=2, y=6, z=14

У наведеному фрагменті функція `duplicate()` отримає три цілих параметри, кожний з яких збільшується вдвічі у тілі функції. Для того щоб ці зміни були відомі у точці виклику (у функції `main()`), всі параметри передаються за посиланнями. Фактичними параметрами є самі змінні `x`, `y` та `z`, а не копії їхніх значень. Функція підставляє змінну `x` замість формального параметра `a`, змінну `y` – замість формального параметра `b`, змінну `z` – замість формального параметра `c`:

```
void duplicate(int& a, int& b, int& c)
               ↑   ↑   ↑
duplicate( x,    y,    z);
```

Розглянемо детальніше різницю поміж передаванням параметра за значенням, за посиланням і за вказівником.

У поданому нижче фрагменті програми функція `f` має три параметри, один з яких передається за значенням, другий – за вказівником, а третій – за посиланням.

Тексти функції та її виклику в основній програмі:

```
#include <iostream>
using namespace std;
void f(int i, int* j, int& k);
int main()
{ int i=1, j = 2, k = 3;
  cout << "i j k\n";
  cout << i << ' ' << j << ' ' << k << '\n';
  f(i,&j,k);
  cout<<i<< ' ' <<j<< ' ' <<k;
  system("pause>>void");
  return 0;
}
void f(int i, int* j, int& k)
{ i++; (*j)++; k++;
}
```

Результат роботи програми:

```
i j k
1 2 3
1 3 4
```

Перший параметр *i* передається до функції за значенням. Його змінення у функції не впливає на вихідне значення. Другий параметр *j* передається за адресою за допомогою вказівника, при цьому для передавання до функції адреси фактичного параметра використовується операція отримання адреси `&j`, а для одержання його значення у функції потрібна операція розіменування (розадресації) `*j`. Третій параметр *k* передається за адресою за допомогою посилання. Це дозволяє передавати фактичний параметр *k* без операції отримання адреси і використовувати його у функції без розадресації.

Якщо виникає потреба уникнути змінювання параметра всередині функції, використовується специфікатор `const`:

```
int f(const char*);
char *t(char *a, const int *b);
```

На місце параметра типу `const&` може передаватися константа, а для змінної за потреби – виконуватися перетворення типу. Вихідні дані, які не повинні змінюватися у функції, більш оптимально є передавати їй за допомогою константних посилань.

Усталено параметри будь-якого типу, окрім масиву й функції, передаються до функції за значенням.

Розглянемо ще один приклад функцій з параметрами, які передаються за посиланням і за вказівником:

```
void swap_values1(float *a, float *b)
{ float temp = *a;
  *a = *b;
  *b = temp;
}
void swap_values2(float &a, float &b)
{ float temp = a;
  a = b;
  b = temp;
}
```

До функції `swap_values1()` параметри передаються за вказівником. Виклик цієї функції може мати вигляд

```
a=3; b=5;
swap_values1(&a, &b);
```

Звертаємо увагу, що фактичними параметрами при передаванні за посиланням і вказівником можуть бути лише змінні.

До функції `swap_values2()` параметри передаються за посиланням. Виклик цієї функції матиме більш природній вигляд:

```
a=3; b=5;
swap_values2(a, b);
```

Якщо функція має повернути понад одне значення, можна передати їй за посиланням додаткові параметри, а у тілі функції їм буде присвоєно значення, які треба повернути. Наступна функція `prevnext()` отримає ціле число `x` й обчислюватиме попереднє `prev` та наступне `next` значення щодо введеного `x`:

```
void prevnext (int x, int& prev, int& next)
{ prev = x-1;
  next = x+1;
}
```

При використуванні *масиву* як параметра функції передається вказівник на його перший елемент, тобто масив завжди передається за адресою (див. п. 5.2.4). При цьому інформація про кількість елементів втрачається і слід передавати його розмірність через окремий параметр. Якщо розмірність масиву є константою, проблем не виникає, оскільки можна зазначити її при оголошенні формального параметра і як верхню межу циклів при опрацюванні масиву всередині функції. У разі передавання до функції масиву символів, тобто рядка, його фактичну довжину можна визначити за розташуванням нуль-символу і передавати кількість елементів немає потреби.

У поданому нижче фрагменті програми оголошується функція `sum()`, параметрами якої є масив і кількість його елементів. Ця функція обчислює суму елементів масиву.

```
int sum (int* mas, int n)
{ // Варіанти: int sum(int mas[ ], int n) або
  // int sum(int mas[n]) (якщо n – константа).
  for (int i = 0, s = 0; i < n; i++) s += mas[i];
  return s;
}
void main()
{ int marks[] = {3, 4, 5, 4, 4};
  cout << "Сума елементів масиву: " << sum(marks, 5);
}
```

При передаванні до функції *багатовимірних масивів* усі розмірності, якщо вони є невідомі на етапі компіляції, мають передаватися як параметри (див. п. 5.3.4). Наприклад, заголовок функції, яка обчислює суму елементів динамічного двовимірного масиву, може мати вигляд:

```
int sum(const int **a, const int nr, const int nc);
```

Виклик цієї функції у програмі може бути таким:

```
cout << "Сума елементів a: " << sum((const int**)a, nstr, nstb);
```

Для звичайного статичного двовимірного масиву, коли обидві розмірності є відомі й є константами, заголовок функції матиме вигляд:

```
int sum(int a[4][6]);
```

Приклад 8.5. Ввести масив з 15 цілих чисел та обчислити суму його парних елементів.

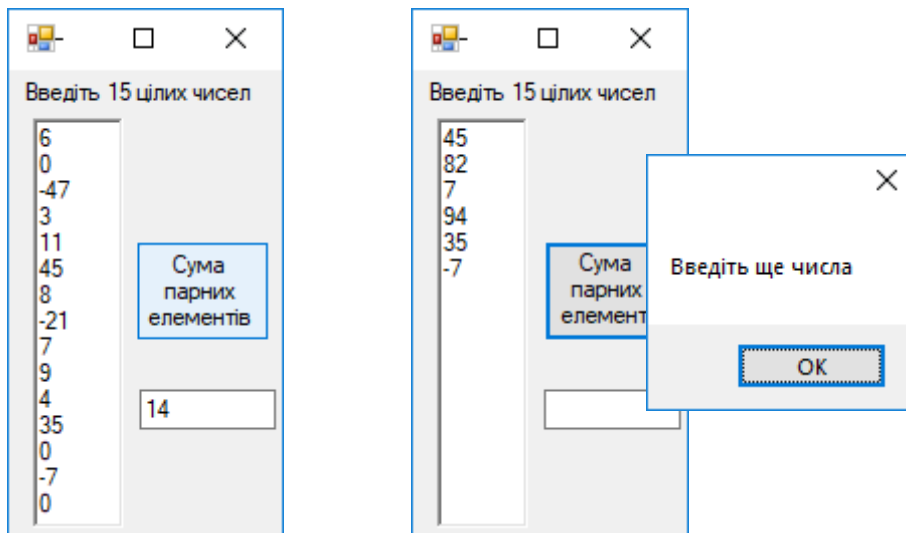
Розв'язок. Функція `sum_par()` має один параметр – масив `A`. Оскільки кількість елементів задано константою, вона є відома і передавати її до функції не

треба. Результат функції має цілий тип `int`.

Тексти функції та основної програми:

```
int sum_par(int A[15])
{ int i, sum;
  for (i=sum=0; i<10; i++)
    if (A[i]%2==0) sum+=A[i];
  return sum;
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
  int i, a[15];
  if (richTextBox1->Lines->Length < 15)
  { MessageBox::Show("Введіть ще числа");
    return;
  }
  for (i=0; i<15; i++) a[i]=Convert::ToInt32(richTextBox1->Lines[i]);
  textBox1->Text=sum_par(a).ToString();
}
```

Результати роботи програми:



Приклад 8.6. Ввести масив й обчислити за допомогою функції добуток від'ємних елементів одновимірного масиву.

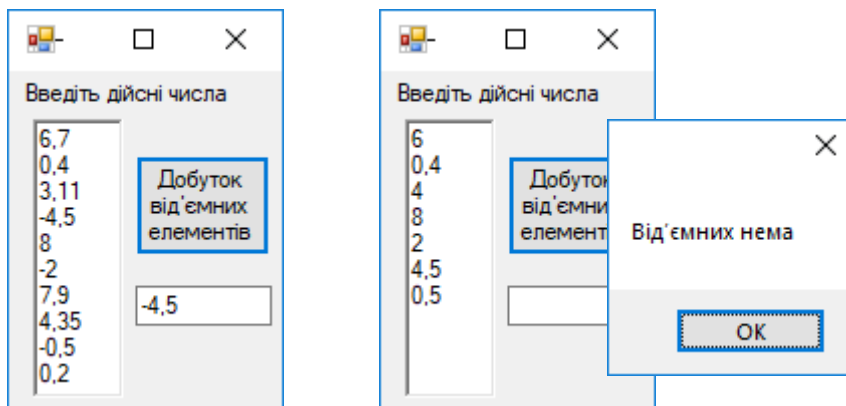
Розв'язок. Оскільки в завданні нічого не сказано про кількість елементів, вважатимемо її довільною. Введемо числа у `richTextBox1` і створимо з них масив. Цей масив буде динамічним, оскільки кількість елементів є заздалегідь невідома.

Функція повертатиме дійсне значення і має два параметри: вказівник на початок масиву і кількість елементів. Якщо від'ємних елементів нема, буде повернуто значення добутка 0.

Тексти функції та її виклику в основній програмі:

```
double dob(double *a, int n)
{ int i, k=0; double p=1;
  for(i=0; i<n; i++)
    if (a[i]<0)
      { k++; p *= a[i]; }
  if (k>0) return p;
  else return 0;
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
  double P; int i, n = richTextBox1->Lines->Length;
  double *a = new double [n];
  for (i=0; i<n; i++) a[i]=Convert::ToDouble(richTextBox1->Lines[i]);
  P = dob(a, n);
  if (P!=0) textBox1->Text = P.ToString(); ;
  else MessageBox::Show("Від'ємних нема");
}
```

Результати роботи програми:



Приклад 8.7. Ввести матрицю дійсних чисел розмірності 4×3 й обчислити за допомогою функції мінімальний елемент матриці.

Тексти функції та її виклику в основній програмі:

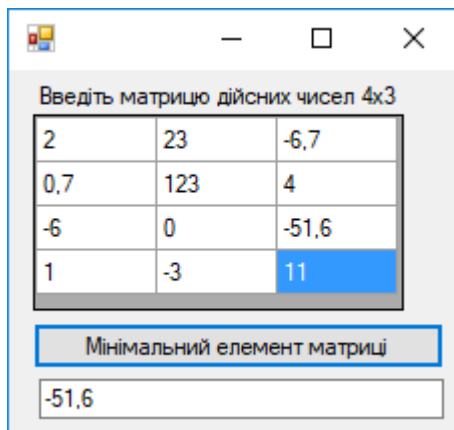
```
double matrix_min(double a[4][3])
{ double min=a[0][0];
  for (int i=0; i<4; i++)
    for (int j=0; j<3; j++)
      if (a[i][j]<min) min=a[i][j];
  return min;
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{ double a[4][3], m;
  for (int i=0; i<4; i++)
    for (int j=0; j<3; j++)
      a[i][j]= Convert::ToDouble(dataGridView1[j,i]->Value);
```

```

m=matrix_min(a);
textBox1->Text = m.ToString();
}
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
    dataGridView1->ColumnCount = 3; // Кількість стовпців
    dataGridView1->Rows->Add(4);    // Кількість рядків
    // Прибрати заголовки стовпців
    dataGridView1->ColumnHeadersVisible = false;
    // Прибрати заголовки рядків
    dataGridView1->RowHeadersVisible = false;
    // Не додавати автоматично рядки
    dataGridView1->AllowUserToAddRows = false;
}

```

Результати роботи програми:



До речі, реалізацію подібного завдання вже було розглянуто в підрозд. 5.3 у прикладі 5.28. До того ж там функція обчислює і повертає три значення: мінімальний елемент матриці та його індекси. При цьому значення мінімального елемента повертається в основну програму через `return`, а обидва індекси передаються за посиланням.

Приклад 8.8. Заповнити матрицю розмірності $M \times N$ (значення M та N увести) випадковими числами з проміжку $[-15, 10]$ за допомогою функції.

Тексти функції та її виклику в основній програмі:

```

void fill_random(int** a, int M, int N)
{
    Random^ rnd = gcnew Random();
    for (int i=0; i<M; i++)
        for (int j=0; j<N; j++) a[i][j]=rnd->Next(-15,11);
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    int M, N, i, j;
    if (textBox1->Text==" " || textBox1->Text=="")
    { MessageBox::Show("Введіть M та N"); return; }
}

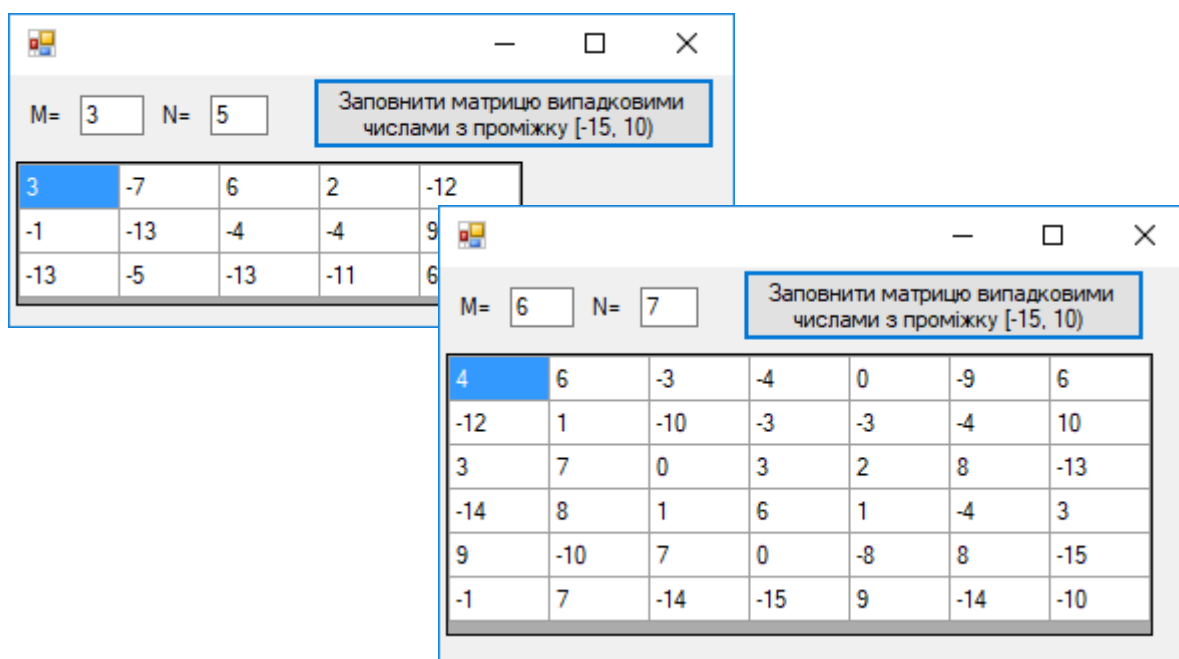
```

```

M=Convert::ToInt32(textBox1->Text);
N=Convert::ToInt32(textBox2->Text);
dataGridView1->Rows->Clear();
dataGridView1->Columns->Clear();
dataGridView1->ColumnCount = N;
dataGridView1->Rows->Add(M);
for (int i=0; i<N; i++) dataGridView1->Columns[i]->Width = 50;
dataGridView1->AllowUserToAddRows = false;
dataGridView1->ColumnHeadersVisible = false;
dataGridView1->RowHeadersVisible = false;
dataGridView1->Size=System::Drawing::Size(3+50*N,3+23*M); // Розміри елемента
if (dataGridView1->Width>=330)
    Form1::Width = dataGridView1->Width+30; // Ширину форми
else Form1::Width = 320;
Form1::Height = dataGridView1->Height+90; // Висота форми
int **a = new int* [M];
for (i=0; i<M; i++) a[i]=new int [N];
fill_random(a, M, N);
for (i=0; i<M; i++)
    for (j=0; j<N; j++) dataGridView1[j,i]->Value=a[i][j].ToString();
for (i=0; i<M; i++) delete []a[i];
delete []a;
}

```

Результати роботи програми:



8.3 Параметри зі значеннями за промовчанням

Щоб спростити виклик функції, в її заголовку можна задати значення параметрів за промовчанням. Ці параметри мають бути останніми у списку й можуть опускатися при виклику функції. Якщо при виклику параметр є опущений,

має бути опущено й усі параметри, що стоять за ним. Як значення параметрів за промовчанням можуть використовуватися константи, глобальні змінні й вирази. Розглянемо приклади прототипів функцій з параметрами за промовчанням.

```
int f(int a, int b=0); // Параметр b має значення за промовчанням 0
void f1(int, int=100, char* = 0); // Зверніть увагу на пробіл між * й =,
// без нього б вийшла операція складного присвоєння *=
void err(int errValue = errno); // errno – глобальна змінна
```

Варіанти виклику цих функцій:

```
f(100); // Виклик функції з першим параметром 100, другий за промовчанням – 0
f(A, 1); // Виклик функції з першим параметром – значенням змінної A, другим – 1
f1(A); // Виклик функції з другим та третім параметрами за промовчанням
f1(a, 10); // Виклик функції зі значенням третього параметра за промовчанням
f1(a, 10, "Hello"); // Виклик без значень параметрів за промовчанням
```

Приклад 8.9. Написати функцію, яка виконує ділення одного цілого числа на друге. Якщо друге число не зазначено, то виконується ділення на 2.

Тексти функції та її виклику в основній програмі:

```
double divide(int a, int b=2)
{ return (double)a/b;
}
void main()
{ cout << divide(12); // 12 : 2
  cout << endl;
  cout << divide(20,4); // 20 : 4
  getch();
}
```

Результати виконання програми:

```
6
5
```

8.4 Функції зі змінною кількістю параметрів

Якщо список формальних параметрів функції завершується трьома крапками, це означає, що при її виклику на цьому місці можна зазначити ще кілька параметрів. Перевірка відповідності типів для цих параметрів не виконується, char та short передаються як int, а float – як double. За приклад стандартної функції зі змінним числом параметрів можна навести функцію printf(), прототип якої має вигляд

```
int printf (const char*, ...)
```

Це означає, що виклик функції має містити принаймні один параметр типу char* й може чи то містити, чи не містити інші параметри:

```
printf("Введіть вихідні дані"); // Один параметр
printf("Сума: %5.2f рублів", sum); // Два параметри
printf("%d %d %d %d", a, b, c, d); // П'ять параметрів
```

Для роботи з функціями зі змінною кількістю аргументів використовуються величини `va_list`, `va_start`, `va_arg` і `va_end`, описані в заголовному файлі `<stdarg.h>`.

Тип `va_list` призначено для зберігання вказівника на черговий аргумент.

Для роботи з функціями зі змінною кількістю аргументів використовуються декілька стандартних макросів (дещо подібних до функцій наборів команд).

Макрос `va_start` ініціює вказівник на черговий елемент, він встановлює аргумент `arg_ptr` на початок списку необов'язкових параметрів і має вигляд функції з двома параметрами:

```
void va_start(arg_ptr, prav_param);
```

де параметр `prav_param` має бути останнім обов'язковим параметром викликуваної функції, а вказівник `arg_ptr` має бути оголошено у списку змінних типу `va_list` у вигляді

```
va_list arg_ptr;
```

Макрос `va_start` має бути використано до першого використання макроса `va_arg`. Макрос `va_arg` повертає значення чергового аргументу, кожен його виклик призводить до просування вказівника, який зберігається в `va_list`. Макрокоманда `va_arg` забезпечує доступ до поточного параметра викликуваної функції й теж має вигляд функції із двома параметрами:

```
type_arg va_arg (arg_ptr, type);
```

Ця макрокоманда бере значення типу `type` за адресою, заданою вказівником `arg_ptr`, збільшує значення вказівника `arg_ptr` на довжину використаного параметра (довжина `type`) і в такий спосіб параметр `arg_ptr` вказуватиме на наступний параметр викликуваної функції. Макрокоманда `va_arg` використовується стільки разів, скільки треба для отримання всіх параметрів викликуваної функції.

Після перебирання аргументів, але до виходу з функції зі змінним числом аргументів, необхідно звернутися до макроса `va_end`. Він встановлює вказівник списку необов'язкових параметрів на `NULL`.

Приклад 8.10. Ввести послідовність цілих чисел й обчислити за допомогою функції зі змінною кількістю параметрів середнє значення цієї послідовності. За завершення послідовності слід вважати значення, яке дорівнює `-1`.

Розв'язок. Оскільки у списку має бути принаймні один елемент, у функції буде один обов'язковий параметр `x`, після якого слід поставити три крапки.

Тексти функції та її виклику в основній програмі:

```
float sred_znach (int x, ...)
{ int i=0, j=0, sum=0;
  va_list uk_arg;
  va_start(uk_arg, x);      // Встановлення вказівника uk_arg на перший
                             // необов'язковий параметр
  if (x!=-1) sum=x;         // Перевірка наявності чисел у списку
  else return 0;
```

```

j++;
while((i=va_arg(uk_arg, int))!=-1) // Вибірка чергового параметра
    { sum+=i; j++; }                // й перевірка завершення списку
va_end(uk_arg);                    //Закриття списку параметрів
return (float)sum/j;
}
void main()
{ float sr=sred_znach(2,3,4,-1);    // Виклик з чотирма параметрами
  cout << "sr=" << sr << endl;
  sr=sred_znach(5,6,7,8,9,-1);      // Виклик із шістьма параметрами
  cout << "sr=" << sr;
  getch();
}

```

8.5 Рекурсивні функції

Рекурсивною називається функція, яка викликає сама себе. Така рекурсія називається прямою. Існує ще непряма рекурсія, коли дві чи більше функцій викликають одна одну.

Несерйозні приклади, які добре ілюструють принцип рекурсії, можна знайти в дитячих лічилках, наприклад:

*У попа був собака, він її любив
 Вона з'їла шматок м'яса, він її убив
 У землю закопав,
 Напис написав:
 «У попа був собака, він її любив
 Вона з'їла шматок м'яса, він її убив
 У землю закопав,
 Напис написав:
 «У попа був собака, він її любив
 Вона з'їла шматок м'яса, він її убив
 У землю закопав,
 Напис написав:
 ... (лапки цитат ніколи не закриваються)*

Загальновідомий приклад рекурсивного зображення – предмет між двома дзеркалами – у кожному з них видно нескінченний ряд відображень.

Окрім того, поняття рекурсії добре ілюструється матрьошками. Це означає, що рекурсія дозволяє звести складне завдання до розв'язку його зменшених копій. Послідовно зменшуючи розмір цих завдань, процес рекурсії в решті решт приведе до відомого розв'язку, використовуючи який можна легко здобути розв'язок початкового завдання.

Більш серйозні приклади рекурсії можна віднайти у математиці: рекурентні співвідношення визначають певний елемент послідовності через один чи кілька попередніх. Наприклад, числа Фібоначчі:

$$F(n) = F(n-1) + F(n-2), \text{ де } F(0)=1, F(1)=1.$$

Якщо розглядати цю послідовність, розпочинаючи від молодших членів до старших, спосіб її побудови задається циклічним алгоритмом, а якщо, навпаки, – від заданого $n=n_0$, то спосіб визначення цього елемента через попередні буде рекурсивним.

Вочевидь, що рекурсія не може бути безумовною, бо у такому разі вона стає нескінченною. Про це свідчить принаймні наведена вище лічилка. **Рекурсія повинна мати всередині себе умову завершення**, за якою наступний крок рекурсії вже не виконуватиметься.

Рекурсивні функції лише на перший погляд схожі на звичайні фрагменти програм. Щоб відчутти специфіку рекурсивної функції, варто простежити за текстом програми перебіг її виконання. У звичайній програмі будемо йти ланцюжком викликів функцій, але жодного разу повторно не увійдемо до того самого фрагмента, допоки з нього не вийшли. Можна стверджувати, що перебіг виконання програми «лягає» однозначно на текст програми. Інша річ – рекурсія. Якщо спробувати відстежити за текстом програми перебіг її виконання, то дістанемось такої ситуації: увійшовши до рекурсивної функції, «рухаємось» її текстом доти, допоки не зустрінемо її виклик (можливо, з іншими параметрами), після чого знову розпочнемо виконувати ту ж саму функцію спочатку. При цьому слід зазначити найважливішу властивість рекурсивної функції: її перший виклик ще не завершився. Чисто зовні складається враження, що текст функції відтворюється (копіюється) щоразу, коли функція сама себе викликає. Насправді цей ефект відтворюється в комп'ютері. Однак копіюється при цьому не весь текст функції (не вся функція), а лише її частини, пов'язані з локальними даними (формальні, фактичні параметри, локальні змінні й точка повертання). Алгоритмічна частина (оператори, вирази) рекурсивної функції й глобальні змінні не змінюються, тому вони присутні в пам'яті комп'ютера в єдиному екземплярі.

Кожний рекурсивний виклик породжує новий «екземпляр» формальних параметрів і локальних змінних, причому старий «екземпляр» не знищується, а зберігається у стеку на засаді вкладеності як «матрьошки». Тут має місце єдиний випадок, коли одному імені змінної в перебігу роботи програми відповідають кілька її екземплярів. Відбувається це в такій послідовності:

- ✓ у стеку резервується місце для формальних параметрів, куди записуються значення фактичних параметрів. Зазвичай це виконується в порядку, зворотному до їхнього місця у списку;
- ✓ при виклику функції до стека записується точка повертання – адреса тієї частини програми, де міститься виклик функції;
- ✓ на початку тіла функції у стеку резервується місце для локальних (автоматичних) змінних.

Зазначені змінні створюють групу (фрейм стека). Стек «пам'ятає історію» рекурсивних викликів у вигляді послідовності (ланцюжка) таких фреймів. Програма у кожний конкретний момент працює з останнім викликом і з останнім фреймом. По завершенні рекурсії програма повертається до попередньої версії рекурсивної функції й до попереднього фрейму у стеку.

Послідовність рекурсивних викликів можна прокоментувати приблизно такою фразою: «Функція F виконує ... і викликає F, яка виконує ... і викликає F...».

Класичним прикладом рекурсивної функції є обчислення факторіала (це не означає, що факторіал треба обчислювати лише в такий спосіб). Для того щоб визначити факторіал числа n , слід помножити n на факторіал числа $(n-1)$:

$n! = n \cdot (n-1)!$ // Це так зване *рекурентне співвідношення*

Відомо також, що $0!=1$ й $1!=1$, тобто умовою зупинки рекурсії буде: якщо $n=0$ чи $n=1$, то факторіал дорівнює 1.

Дамо повне рекурсивне визначення факторіала:

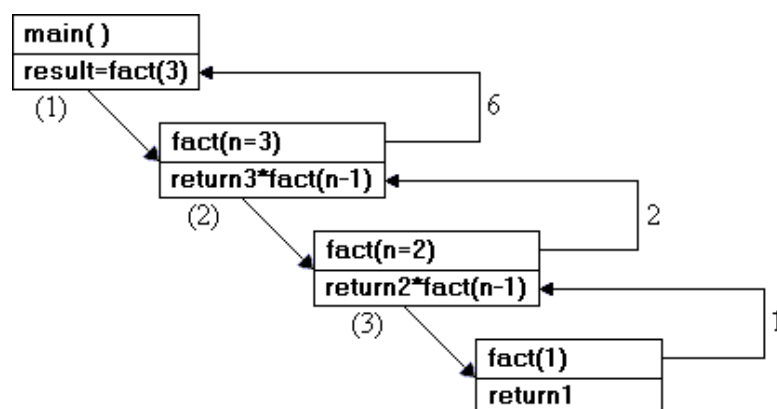
$$n! = \begin{cases} (n-1)! \cdot n, & n > 0; \\ 1, & n = 0. \end{cases}$$

Отже, рекурсивна функція обчислення факторіала числа n виглядатиме так:

```
long fact (long n)
{ if (n==0 || n==1) return 1;
  return (n*fact(n-1));
}
```

Якщо у функції `main()` зустрінеться виклик функції `fact(3)`, при виконванні цієї функції буде викликано функцію `fact(2)`, яка своєю чергою викличе `fact(1)`. Для останньої спрацює умова зупинки рекурсії ($n==1$) і у функцію `fact(2)` буде повернуто значення 1 та обчислено значення 2 ($2! = 2$), яке своєю чергою буде повернуто до функції `fact(3)` і буде використано для обчислення $3!$.

Отже, програма має 3 рекурсивні виклики. Занурившись на три рівні рекурсії, програма дісталася «глухого кута» – граничної точки, після чого вона зворотним ланцюжком має піднятися на ті самі три рівні. В результаті кожного рекурсивного виклику функція зберігає своє значення змінної n , оскільки до третього виклику у неї буде вже три окремих змінних з ім'ям n , при цьому кожна має власне, відмінне від інших, значення, однак тіло функції буде присутнім в пам'яті в єдиному екземплярі. Цей ланцюжок рекурсивних викликів можна зобразити в такий спосіб:



Породження усіх нових копій рекурсивної функції до виходу на граничну умову називається *рекурсивним спуском*. Максимальна кількість копій рекурсивної функції, яке водночас може міститися у пам'яті комп'ютера, називається *глибиною рекурсії*. У разі відсутності граничної умови необмежене зростання кількості таких копій призведе до аварійного завершення програми за рахунок

переповнення стека. Завершення роботи рекурсивних функцій, аж до самої першої, яка ініціювала рекурсивні виклики, називається *рекурсивним підйомом*.

При створюванні рекурсивної функції завжди слід добре поміркувати, чи ефективно є застосовувати рекурсивний алгоритм, оскільки рекурсія з надто великою кількістю вкладених викликів може вичерпати ресурси оперативної пам'яті, що спричинить збій у роботі операційної системи.

Розглянемо алгоритм нераціонального використання рекурсії на прикладі обчислення чисел Фібоначчі – числового ряду, в якому кожен наступний член є сумою двох попередніх: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 тощо. Італійський математик середньовіччя Леонардо Фібоначчі відкрив цю числову послідовність, вивчаючи піраміду Хеопса у Гізі. До речі, ці числа пов'язані поміж собою цікавими співвідношеннями. Наприклад, кожне число є приблизно в 1.618 разів більше попереднього (цю пропорцію називають золотую), у 2.618 рази більше того, яке розташоване через одне число від нього і в 4.236 разів більше числа, розміщеного двома числами раніш, а кожне попереднє число складає приблизно 0.618 від наступного. Поширеність співвідношень Фібоначчі у житті просто вражає. Принцип золотого перетину – найвищий прояв структурної й функціональної довершеності цілого та його частин у мистецтві, науці, техніці й природі. Закономірності «золотої» симетрії проявляються в енергетичних переходах елементарних часток, у будові деяких хімічних сполук, у планетарних і космічних системах, у генних структурах живих організмів, у принципах формотворення рослин тощо. Ці закономірності, виявлено як у будові окремих органів людини, так і тіла у цілому, а також вони проявляються у біоритмах і функціонуванні головного мозку і зорового сприйняття.

Ітераційна функція обчислення чисел Фібоначчі має вигляд

```
long fib1(int n)
{ long a=0, b=1, c=n;
  for(int i=2; i<=n; i++)
    { c=a+b;  a=b;    b=c;  }
  return c;
}
```

Рекурсивна функція матиме вигляд

```
long fib2(int n)
{ if (n==0 || n==1) return n;
  else return fib2(n-1)+fib2(n-2);
}
```

або

```
long fib3(int n)
{ return (n>1) ? fib3(n-1)+fib3(n-2) : n;
}
```

Така рекурсивна функція є придатна для використання лише для невеликих значень n , оскільки кожне звертання призводить до двох подальших викликів, причому викликів з одним і тим самим значенням параметра, тобто кількість викликів зростає експоненціально. Отже, у цьому завданні доцільніше застосовувати ітераційну функцію **fib1**.

Рекурсивні функції найчастіше застосовують для компактної реалізації рекурсивних алгоритмів, а також для роботи зі структурами даних, описаними рекурсивно, наприклад, з бінарними деревами. Кожну рекурсивну функцію можна реалізувати без застосування рекурсії. Достоїнством рекурсії є компактний запис, а недоліками – витрата часу й пам'яті на повторні виклики функції й передавання їй копій параметрів і, головне, небезпека переповнення стека.

Приклад 8.11. Ввести ціле число і вивести всі цифри, які зображують це число.

Текст консольної програми:

```
void cnum (int n)
{ int a=10;
  if (n==0) return;
  else { cnum(n/a);   cout << n%a << "\t";  }
}
void main()
{ int n;
  cout << "Ввести ціле число: ";   cin >> n;
  cnum(n);
  getch();
  return 0;
}
```

Результати виконання програми:

```
Ввести ціле число: 1234
1      2      3      4
```

Приклад 8.12. Створити рекурсивну функцію піднесення числа до степеня. Аргументами цієї функції мають бути певне дійсне число, яке буде підноситись до степеня, і, безпосередньо, показник степеня цілого типу.

Тексти функції та її виклику в основній програмі:

```
#include <iostream.h>
#include <conio.h>
double power(double x, int st)
{ if (st<=0) return 1;
  else return(x*power(x,st-1));
}
void main()
{ double x; int st;
  cout<<"Ввести число x= ";   cin>>x;
  cout<<"Ввести показник степеня st= "; cin>>st;
  cout<<"x ^ st= " << power(x, st);
  getch(); return 0;
}
```

Результати виконання програми:

```
Ввести число x= 2.5
```

Ввести показник степеня $st=2$
 $x^{st}=6.25$

Приклад 8.13. Написати рекурсивну функцію, яка обчислює мінімальний елемент масиву.

Розв'язок. Створимо рекурсивну функцію `minimum()`, яка повертатиме індекс мінімального елемента. Вона матиме три параметри: 1) вказівник на початок масиву `mas`, 2) розмір масиву `n` і 3) індекс елемента, з якого розпочинається підмасив. Цей підмасив рекурсивно зменшується, допоки його розмір не становитиме 1. Коли у ньому залишиться один елемент ($k==n-1$), – повертається значення `k`. На наступному кроці рекурсивного повертання порівнюються `mas[k]` (останній елемент) та `mas[a]` (передостанній елемент) та повертається індекс меншого з них. Продовжуючи, на кожному кроці порівнюються мінімум, здобутий на попередніх кроках, і `mas[a]` (поточний елемент).

Тексти функції та її виклику в основній програмі:

```
int minimum(int *mas, int n, int k)
{ if(k == n-1) return k;
  int a = minimum(mas,n,k+1); // Виклик для підмасиву розміром на 1 менше
  if (mas[a]<mas[k]) return a;  else return k;
}
void main()
{ int i, n, min;
  cout<<"Введіть розмір масиву: ";   cin>>n;
  int *mas = new int[n];
  for (i=0;i<n;i++)
  { cout<<"Введіть mas["<<i<<"]: ";   cin>>mas[i]; }
  cout<<endl;
  min = minimum(mas,n,0);           // Виклик для всього масиву (k=0)
  cout<<"Мінімум: mas["<<min<<" = "<<mas[min]<<endl;
  getch();
  return 0;
}
```

Приклад 8.14. Написати рекурсивну функцію, яка обчислює суму квадратних коренів натуральних чисел від 1 до `n`.

Тексти функції та її виклику в основній програмі:

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
double sumkor(int a)
{ if (a==1) return 1;
  else return (sqrt(a) + sumkor(a-1));
}
void main()
{ int n;
  cout<<"n= ";   cin>>n;
```

```

    cout<<sumkor(n);
    getch(); return 0;
}

```

Приклад 8.15. Ввести масив з 10-ти дійсних чисел і створити рекурсивну функцію, яка записує масив у зворотній послідовності .

Розв'язок. Функція `invertItem()` змінює місцями елементи з індексами `i` та `j`. Спочатку це перший та останній елементи масиву, потім індекс `i` збільшується, а `j` – зменшується – і функція викликає себе для обміну другого й передостаннього елементів. Ланцюжок викликів триває, допоки `i < j`.

Тексти функції та її виклику в основній програмі:

```

void invertItem(double *a, int i, int j)
{ double t=a[i]; a[i]=a[j]; a[j]=t;
  i++; j--;
  if (i<j) invertItem(a, i, j);
}
void main()
{ double a[10]; int i;
  cout<<"Ввести 10 дійсних чисел:"<<endl;
  for (i=0; i<10; i++) cin>>a[i];
  invertItem(a, 0, 9);
  for (i=0; i<10; i++) cout<<a[i]<<" ";
  getch(); return 0;
}

```

Результати виконання програми:

```

Ввести 10 дійсних чисел:
0 1 2 3 4 5 6 7 8 9
9 8 7 6 5 4 3 2 1 0

```

Приклад 8.16. Визначити, чи є введений рядок паліндромом, тобто таким, що читається однаково зліва направо і справа наліво.

Розв'язок. Подібна задача розглядалась у прикладі 7.51 для рядків типу `wstring`. При чому там перевірка реалізовувалась без урахування пробілів і знаків пунктуації.

Тексти функції та її виклику в основній програмі:

```

#include <iostream>
using namespace std;
int pal(char *s)
{
  int len;
  char *s1=new char[strlen(s)+1];
  if (strlen(s)<=1) return 1;
  else
  { len=(s[0]==s[strlen(s)-1]);
    strncpy(s1, s+1, strlen(s)-2);

```

```

    s1[strlen(s)-2]='\0';
    return len && pal(s1);
}
delete []s1;
}
int main()
{
    char s[100];
    setlocale(0, ".1251"); cout<<"Введіть рядок: "<<endl;
    setlocale(0, ".OCP"); cin>>s;
    int p = pal(s);
    setlocale(0, ".1251");
    if (p) cout<<"Рядок є паліндромом"<<endl;
    else cout<<"Рядок не є паліндромом"<<endl;
    system("pause>>void");
    return 0;
}

```

Зверніть увагу на команду:

```
len=(s[0]==s[strlen(s)-1]);
```

У ній спочатку виконується порівняння $s[0]==s[strlen(s)-1]$, а потім його результат (true чи false) перетворюється на ціле число (відповідно 1 чи 0) і присвоюється змінній `len`.

Запис

```
return 1 && pal(s1);
```

означає, що після обчислення `pal(s1)` обчислюється результат операції логічного множення `&&` і цей результат повертається у точку виклику функції.

Результати виконання програми:

Введіть рядок:

шалаш

Рядок не є паліндромом

Приклад 8.17. Ввести два числа й обчислити найбільший спільний дільник (НСД) двох цілих чисел, застосовуючи рекурсивний алгоритм Евкліда.

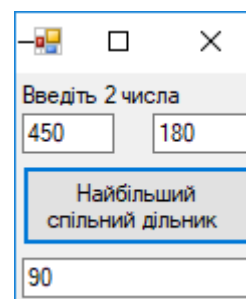
Розв'язок. Нерекурсивна реалізація алгоритму Евкліда обчислення НСД двох цілих чисел наведена у п. 4.3.4.

Тексти функції та основної програми:

```

int nod(int a, int b)
{ int c;
  // За умови b>a параметри змінюються місцями
  if (b > a) c = nod(b, a);
  else
    if (b == 0) c = a;
    else c = nod(b, a%b); // Тут a%b – залишок від ді-
лення a на b

```




```
    return c;
}
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    int a, b;
    a = Convert::ToInt32(textBox1->Text);
    b = Convert::ToInt32(textBox2->Text);
    textBox3->Text = nod(a, b).ToString();
}
```

Приклад 8.18. Ввести масив з 12-ти цілих чисел і упорядкувати їх за зростанням методом швидкого сортування за допомогою рекурсивної функції.

Розв'язок. Швидке сортування (англ. quicksort), часто зване qsort на ім'я реалізації в стандартній бібліотеці мови C – широко відомий алгоритм сортування, розроблений англійським інформатиком Чарльзом Хоаром. Це один зі швидких відомих універсальних алгоритмів сортування масивів.

Коротко описуючи цей алгоритм, треба вибрати опорний елемент, порівняти решту елементів з опорним, на підставі порівняння розбити множину елементів масиву на три – «менші опорного», «рівні» і «більші», розташувати їх у порядку менші-рівні-більші, після чого повторити рекурсивно ці дії для «менших» і «більших».

Зазвичай опорним елементом вибирають середній елемент, хоча з точки зору коректності алгоритму це не є суттєвим. Операція реорганізації масиву полягає в тому, щоб усі елементи, які є менші чи рівні опорному елементові, опинилися ліворуч від нього, а усі елементи, більші за опорний, – праворуч від нього. Для цього у функції QuickSort() організовано таку послідовність дій:

1) два індекси – L та R, прирівнюються до мінімального та максимального індексів розділеного масиву відповідно: $i=L$; $j=R$;

2) обчислюється індекс опорного елемента $n=(L+R)/2$;

3) індекс i послідовно збільшується допоки i-й елемент не перевищить опорний: `while (a[i] < x) i++`;

4) індекс j послідовно зменшується допоки j-й елемент не виявиться менше чи рівний опорному: `while (x < a[j]) j--`;

5) якщо $i < j$ – знайдену пару елементів треба поміняти місцями й продовжити операцію поділу з тих значень i та j, які були досягнені;

6) чергуючи зменшення j та збільшення i, порівняння та необхідні обміни, віднаходимо середину масиву, коли $i=j$, тобто операцію поділу завершено, обидва індекси вказують на опорний елемент. У результаті елементи масиву виявляються розділеними на дві частини так, що всі елементи ліворуч є менші за опорний елемент, а всі елементи праворуч – більше за нього;

7) рекурсивно впорядковуємо підмасиви, що лежать ліворуч і праворуч від опорного елемента, застосовуючи розглянутий алгоритм. Оскільки на кожній ітерації (на кожному наступному рівні рекурсії) довжина оброблюваного відрізка масиву зменшується, щонайменше, на одиницю, термінальна гілка рекурсії буде досягнена завжди й опрацювання гарантовано завершиться.

Цікаво, що Хоар розробив цей метод для машинного перекладу: справа в тім, що в той час словник зберігався на магнітній стрічці, і якщо впорядкувати всі слова в тексті, їх переклади можна дістати за один прогін стрічки.

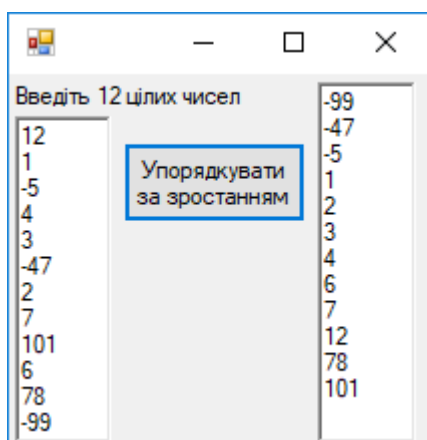
Тексти функції та її виклику в основній програмі:

void QuickSort(int a[], int L, int R) // *L – ліва та R – права межі сортування*

```
{ int i=L, j=R, x, y;
  int n=(L+R)/2;
  x = a[n];
  do
  { while (a[i] < x) i++;
    while (x < a[j]) j--;
    if ( i <= j )
    { y=a[i];
      a[i]=a[j];
      a[j]=y;
      i++; j--;
    }
  } while (i <= j);
  if (L < j) QuickSort(a, L, j);
  if (i < R) QuickSort(a, i, R);
  return;
}
```

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
  int i, a[12];
  for (i=0; i<12; i++) a[i]=Convert::ToInt32(richTextBox1->Lines[i]);
  QuickSort(a, 0, 11 ); // На вході ліва і права межа сортування
  richTextBox2->Clear();
  for (i=0; i<12; i++) richTextBox2->Text+=a[i].ToString()+"\n";
}
```

Результати виконання програми:



8.6 Перевантаження функцій

Часто буває зручно, щоб функції, які реалізують однаковий алгоритм для різних типів даних, мали однакове ім'я. Якщо це ім'я несе певну інформацію, це робить програму більш зрозумілою, оскільки для кожної дії треба пам'ятати лише одне ім'я. Використання кількох функцій з однаковим ім'ям, але з різними типами параметрів називається *перевантаженням* функцій.

Компілятор визначає, яку саме функцію слід викликати, за типом фактичних параметрів. Цей процес називається *дозволом перевантаження* (переклад з англійського слова «resolution»). Тип значення, яке повертає функція, у дозволі не бере участі. Механізм дозволу ґрунтується на доволі складному наборі правил, зміст яких зводиться до того, щоб використати функцію з найбільш придатними аргументами й видати повідомлення, якщо така не віднайдеться. Мета перевантаження – зробити так, щоб функції з однаковим ім'ям виконувалися по-різному (і за можливість повертали різні значення) при звертанні до них з різними за типами й кількістю параметрами. За приклад наведемо чотири варіанти функції для визначення максимального значення:

```
int max(int, int);           // Повертає найбільше з двох цілих
char* max(char*, char*);    // Повертає рядок максимальної довжини
// Повертає найбільше значення з першого параметра і довжини рядка,
// переданого до функції в якості другого параметра
int max(int, char*);
// Повертає найбільше значення з першого параметра і довжини рядка,
// переданого до функції в якості першого параметра
int max(char*, int);
void f(int a, int b, char* c, char* d)
{ cout << max(a,b) << max(c,d) << max(a,c) << max(c,b);
}
```

Усі чотири функції мають однакове ім'я (max) й відрізняються типами параметрів (тому вважаються за різні функції). За викликання функції max() компілятор вибирає відповідно до типу фактичних параметрів варіант функції (у наведеному прикладі буде послідовно викликано всі чотири варіанти функції). Якщо точної відповідності не віднайдено, виконуються перетворювання порядкових типів відповідно до загальних правил, наприклад, `bool` й `char` до `int`, `float` до `double` тощо. Далі виконуються стандартні перетворювання типів, наприклад, `int` до `double`, вказівників до `void*` тощо. Наступним кроком є виконання перетворювань типів, заданих користувачем, а також пошук відповідностей за рахунок змінної кількості аргументів функції. Якщо відповідність на тому самому етапі може бути набута у понад один спосіб, виклик вважається за неоднозначний і видається повідомлення про помилку.

Неоднозначність може виникнути за:

- ✓ перетворення типу;
- ✓ використання параметрів-посилань;
- ✓ використання аргументів за промовчанням.

Приклад неоднозначності за перетворення типу:

```
#include <iostream>
using namespace std;
int f(int a)
{ return a; }
int f(int a, int b=1)
{ return a*b; }
int main ()
{ cout << f(10,2); // Викликається f(int, int)
  cout << f(10);   // Неоднозначність – що викликати: f(int, int) чи f(int)?
  return 0;
}
```

Слід відрізнити перевантаження функцій і редекларацію. Якщо функції з однаковим ім'ям мають однакові параметри і типи результату, вважається, що друге визначення розглядатиметься як редекларація (повторне визначення). Якщо параметри функцій з однаковим ім'ям збігаються і функцій відрізняються лише типом значення, яке повертається, це призведе до помилки.

Правила опису перевантажених функцій:

- ✓ перевантажені функції мають перебувати в одній області видимості, інакше станеться приховування аналогічне до однакових імен змінних у вкладених блоках;

- ✓ перевантажені функції можуть мати параметри за промовчанням, при цьому значення одного й того самого параметра у різних функціях мають збігатися. У різних варіантах перевантажених функцій може бути різна кількість параметрів за промовчанням.

Функції не можуть бути перевантажені, якщо опис їхніх параметрів відрізняється лише специфікатором `const` чи використанням посилання, наприклад, `int` та `const int` чи `int` та `int&`.

Приклад 8.19. Написати дві функції для обчислення суми сімох елементів цілого й дійсного масивів відповідно. Використати ці функції для обчислення сум двох масивів.

Тексти функцій та їх викликів в основній програмі:

```
#include <iostream>
using namespace std;

int adder(int iarray[7])
{ int isum = 0;
  for (int i=0; i<7; i++) isum += iarray[i];
  return isum;
}

double adder(double farray[7])
{ double fsum = 0;
  for (int i=0; i<7; i++)
    fsum += farray[i];
  return fsum;
}
```

```

int main()
{ int iarray[7] = {5,1,6,20,15,0,12};
  double farray[7] = {3.3,5.2,0.05,1.49,3.12345,31.0,2.007};
  int isum; double fsum;
  isum = adder (iarray);      // Виклик для масиву цілих чисел
  fsum = adder (farray);      // Виклик для масиву дійсних чисел
  setlocale(0, ".1251");
  cout << "Сума масиву цілих чисел дорівнює " << isum << "\n";
  cout << "Сума масиву дійсних чисел дорівнює " << fsum << "\n";
  cin.get();  return 0;
}

```

Результати виконання програми:

Сума масиву цілих чисел дорівнює 59
 Сума масиву дійсних чисел дорівнює 46.1705

Приклад 8.20. Написати три функції для обчислення периметра трикутника, чотирикутника й п'ятикутника відповідно.

Розв'язок. Завдання відрізняється від попереднього тим, що тип параметрів є однаковий, а кількість параметрів – різна. При викликанні функції вибирається той варіант, для якого збігається кількість фактичних і формальних параметрів.

Тексти функції та її виклику в основній програмі:

```

#include <iostream>
using namespace std;
// Визначення всіх потрібних версій функції
// Версія функції для трикутника
int Perimetr(int Len1, int Len2, int Len3)
{ return Len1 + Len2 + Len3; }
// Версія функції для чотирикутника
int Perimetr(int Len1, int Len2, int Len3, int Len4)
{ return Len1 + Len2 + Len3 + Len4; }
// Версія функції для п'ятикутника
int Perimetr(int Len1, int Len2, int Len3, int Len4, int Len5)
{ return Len1 + Len2 + Len3 + Len4 + Len5; }
int main()
{ int AB = 5, BC = 3, CA = 7, CD = 8, DA = 11, DE = 2, EA = 9;
  cout << "Perimetr ABC = " << Perimetr(AB, BC, CA) << "\n";
  cout << "Perimetr ABCD = " << Perimetr(AB,BC,CD,DA) << "\n";
  cout << "Perimetr ABCDE = " << Perimetr(AB, BC, CD, DE, EA);
  cin.get();  return 0;
}

```

Результати виконання програми:

Perimetr ABC = 15
 Perimetr ABCD = 27
 Perimetr ABCDE = 27

Питання та завдання для самоконтролю

- 1) Що таке прототип функції?
- 2) У чому полягає різниця поміж оголошенням прототипу й визначенням функції?
- 3) Які дії виконує у функції оператор `return`?
- 4) Для яких функцій при їхньому визначенні не використовується ключове слово `return`?
- 5) Який тип має функція, яка не повертає значення?
- 6) Яку інформацію про функцію надає такий її заголовок:

```
int fun(double x)
```
- 7) В який спосіб можна передавати з функції декілька результатів?
- 8) Що означає параметр функції, який передається за адресою за допомогою посилання?
- 9) Які параметри функції називають формальними, а які – фактичними?
- 10) Чи повинні збігатися імена відповідних формальних і фактичних параметрів функції?
- 11) Коли при оголошуванні змінних у функціях слід використовувати специфікатор `static`?
- 12) Що таке локальна змінна?
- 13) Пояснити різницю поміж локальними і глобальними змінними.
- 14) В яких випадках слід використовувати глобальні змінні?
- 15) Що буде виведено на екран після виконання такої програми?

```
#include <iostream.h>
void swap(int i, int j)
{ int temp = i; i = j; j = temp;
}
int main()
{ int a = 10, b = 3;
  swap (a, b);
  cout << "a = " << a << " and b = " << b;
  return 0;
}
```
- 16) Що таке область видимості змінної?
- 17) Чи можна у C++ оголошувати вкладені функції?
- 18) Які є способи передавання параметрів до функції?
- 19) Що буде виведено на екран після виконання програми у завданні 12, якщо замінити функцію `swap()` на таку:

```
void swap(int &i, int &j)
{ int temp = i; i = j; j = temp; }
```
- 20) Що таке перевантаження функцій?
- 21) Назвати правила опису перевантажених функцій.
- 22) Яка проблема виникне з такими перевантаженими функціями?

```
void inc(int &i)
{ i = i + 1; }
```

```
void inc(int &i, int diff = 1)
{ i += diff;
}
```

23) Знайти помилку у функції

```
double volume(double length, double width = 1, double height)
{ return length * width * height;
}
```

24) Пояснити спосіб передавання параметрів до функції

```
void inc (int &i, int diff = 1)
{ i += diff;
}
```

25) Яка помилка припущено у такій програмі? Як її виправити?

```
#include <iostream.h>
int main()
{ double x = 5.2;
  cout << x << " ^ 2 = " << sqr(x);
  return 0;
}
double sqr(double x)
{ return x * x;
}
```

26) Що таке рекурсія?

27) Охарактеризувати послідовність, в якій відбувається кожний рекурсивний виклик?

28) Що називається рекурсивним підйомом?

29) Чи є наступна рекурсивна функція правильною? Пояснити.

```
double F(double x)
{ return F(x-1);
}
```

30) Написати програму із застосуванням перевантаження функцій для обчислення суми двох чисел цілого і дійсного типів відповідно.

РОЗДІЛ 9

БАГАТОФАЙЛОВІ ПРОГРАМИ

9.1 Міжфайлова взаємодія

Проект Visual C++ складається з багатьох файлів, але власне програма тепер містилася в єдиному .cpp файлі (наприклад, у файлі ConsoleApplication.cpp). Існує багато ситуацій у програмуванні, коли мати лише один файл програми незручно.

Якщо певний проект розробляє група програмістів, кожен з яких виконує окреме завдання, має сенс розв'язок кожного завдання розташовувати в окремому файлі, а потім долучати всі такі файли до спільного проекту. Такий підхід дозволяє більш оптимально організувати роботу команди програмістів і більш чітко визначити взаємодію частин програми. Відповідно до засад технології програмування, великі програми розбиваються на частини за функціональною направленістю. Один файл може містити програмний код підтримки графічного виведення, другий – математичні розрахунки, третій – код введення-виведення даних файлу. При написанні великих програм тримати всі ці частини в одному файлі-програмі складно чи взагалі неможливо.

Іноді виникає потреба у використуванні в різних програмах однакових типів і функцій. В таких ситуаціях є сенс винести оголошення цих типів і функцій до окремого файлу й долучати його за потреби до різних програм. Має сенс створювати окремі бібліотеки класів і розміщувати їх в окремих файлах.

У багатофайлових програмах елементи програми (змінні, константи, функції тощо), які зберігаються у різних файлах, повинні взаємодіяти один з одним. Розглянемо спочатку окремі .cpp-файли. Зазначимо, що *область видимості* – це частина програми, всередині якої до певної змінної чи іншого елемента програми є доступ (детальніше див. підрозд. 9.4). Наприклад, елементи, оголошені всередині функції, мають *локальну область видимості*, тобто доступ до них існує лише з даної функції. Компоненти класу є видимими лише всередині класу, окрім певних окремих випадків.

Елементи програми, оголошені зовні усіх функцій та класів, мають *глобальну область видимості*, тобто доступ до них можливий з кожного місця коду програми. Окрім того, глобальні елементи є видимі і з інших файлів проекту.

Уточнимо різницю поміж *оголошенням* та *визначенням* змінних у багатофайлових проектах. При *оголошенні* задаються тип та ім'я змінної. Це не означає, що для неї одразу ж зарезервується місце у пам'яті. Оголошення лише повідомляє компіляторіві, що десь у програмі може зустрітися змінна такого типу з таким ім'ям. Змінна *визначається*, коли для неї зарезервовується місце у пам'яті, яке здатне містити її значення. Більшість оголошень є визначеннями. Єдиним оголошенням простої змінної, яке не є визначенням, є оголошення з використанням зарезервованого слова **extern** (без ініціалізації):

```
int a;           // Оголошення і визначення
```

```
extern int b;    // Лише оголошення
```

Глобальну змінну може бути визначено у програмі лише одноразово, незалежно від того, скільки файлів міститься у програмі. Локальні змінні з однаковими іменами може бути визначено у різних областях видимості. Але використовувати змінні з однаковими іменами, навіть у різних блоках програми, не рекомендовано, оскільки це може призвести до помилок.

Щоб глобальна змінна, визначена в одному файлі, була видимою в усіх файлах програми, слід оголосити її із зарезервованим словом `extern`:

```
// Файл A
int X;           // Визначення змінної X у файлі A
// Файл B
extern int X;     // Визначення змінної X у файлі B
X=3;
```

У цьому прикладі оголошення змінної `X` у файлі `A` зробило її видимою у файлі `B`. Зарезервоване слово `extern` означає, що оголошення в такому разі – це лише оголошення, й нічого більше. Воно спонукає компілятор, який кожного моменту часу бачить лише один з файлів, не звертати уваги на те, що змінна `X` не є визначена у файлі `B`.

Слід пам'ятати про одне обмеження: змінну не можна ініціалізувати при оголошенні з `extern`. Вираз

```
extern int X=27;
```

примусить компілятор вважати, що це не є оголошення, а визначення. Тобто він проігнорує слово `extern` і сприйме оголошення за визначення. Якщо цю змінну в іншому файлі вже визначено, буде видано помилку повторного визначення.

Якщо треба мати дві глобальні змінні з однаковими іменами у двох різних файлах, їх слід визначати із зарезервованим словом `static`. У такий спосіб область видимості змінної звужується до файлу, в якому її визначено. Решта змінних з таким ім'ям можуть в інших файлах використовуватися без обмежень.

```
// Файл A
static int X; // Визначення: змінна X є видима лише у файлі A
// Файл B
static int X; // Визначення: змінна X є видима лише у файлі B
```

Хоча тут визначено дві змінні з однаковими іменами, конфлікту це не спричинює. Код, до якого входить звертання до `X`, звертатиметься лише до змінної, визначеної у даному файлі. У такому разі вважають, що статична змінна має *внутрішнє зв'язування*. Нестатичні глобальні змінні мають *зовнішнє зв'язування*.

У багатофайлових програмах рекомендовано глобальні змінні робити статичними, якщо за логікою роботи доступ до них вимагається не більше ніж з одного файлу. Це допоможе запобігти виникненню помилки, пов'язаної з випадковим задаванням того ж самого імені в іншому файлі. Окрім того, текст програми стає більш прозорим – не треба піклуватися про збіг імен у різних файлах.

Звернімо увагу на те, що зарезервоване слово `static` має декілька значень залежно від контексту. У контексті глобальних змінних слово `static` звужує область видимості змінної до одного файлу.

Змінна-константа, яка визначається за допомогою `const`, у загальному випадку є невидимою за межами одного файлу. В цьому вона є подібна до `static`. Але її можна зробити видимою з кожного файлу програми і для визначення, і для оголошення за допомогою слова `extern`:

```
// Файл А
extern const int X=99;    // Визначення X
// Файл В
extern const int X;       // Оголошення X
```

Тут файл В матиме доступ до константи X, визначеної в А. Компілятор розрізняє оголошення й визначення константи за наявністю чи то відсутністю ініціалізації.

Оголошення функції задає її ім'я, тип значення, яке повертає функція, і типи всіх аргументів. *Визначення функції* – це оголошення плюс реалізація функції (код, який міститься всередині фігурних дужок).

Коли компілятор генерує виклик функції, йому не треба знати, в який спосіб вона працює. Все, що потрібно знати, – це її ім'я, тип результату і типи параметрів (аргументів). Усі дані стосовно цього містяться в оголошенні функції. Тому можна легко визначити функцію в одному файлі, а викликати її з іншого. Жодних додаткових дозволів (як-от слова `extern`) не вимагається. Слід лише оголосити функцію в тому файлі, звідки ця функція викликатиметься.

```
// Файл А
int add(int a, int b)      // Визначення функції
{ return a+b; }           // з тілом функції
// Файл В
int add(int, int);         // Оголошення функції (без тіла)
...
int answer=add(2, 3);      // Виклик функції
```

Зарезервоване слово `extern` з функціями не використовується, оскільки компілятор здатен відрізнити оголошення від визначення за наявності чи відсутності тіла функції.

9.2 Заголовні файли

Заголовні файли мають розширення `h` і містять заголовки (прототипи) функцій, а також оголошення типів, констант та змінних з ключовим словом `extern`. Перевага заголовних файлів полягає у тому, що, коли створено заголовний файл для одного модуля (одного чи декількох `сpp`-файлів) програми, можна переносити всі зроблені оголошення, які були зроблені, до іншої програми C++. Для цього слід залучити заголовний файл за допомогою директиви `#include`. За звичай для кожного `сpp`-файлу створюється власний заголовний файл з таким самим ім'ям і розширенням `h`. І навпаки, здебільшого для кожного заголовного файлу слід створити `сpp`-файл з реалізацією функцій, оголошених у `h`-файлі.

При залученні до програми заголовного файлу його текст (а з ним і текст відповідного `сpp`-файлу) автоматично вставляється до програми замість рядка з відповідною директивою `#include`. При компіляції програми всі залучені заго-

ловні файли з файлами їхньої реалізації перекомпільовуються, тому залучення надмірної кількості заголовних файлів уповільнює компіляцію програми.

Заголовні файли можна розділити на стандартні й створювані програмістом. Стандартні заголовні файли зберігаються у спеціальній папці INCLUDE, яка розташована у папці, в яку було встановлено Visual Studio. Імена стандартних заголовних файлів пишуться у кутових дужках <>, наприклад:

```
#include <math.h>      /* Долучення заголовного файлу
                        з математичними функціями */
```

Окрім того, для таких файлів можна не зазначати розширення .h. Якщо стандартний заголовний файл розміщено в іншій папці, слід зазначити його місцезнаходження, наприклад, щоб залучити до програми заголовний файл types.h, який міститься у папці INCLUDE\SYS, слід написати:

```
#include <include\sys\types.h>
```

Заголовний файл може містити:

- ✓ оголошення типів `typedef double arr [14];`
- ✓ шаблони типів `template <class T>`
`class V { /* ... */ }`
- ✓ оголошення (прототипи) функцій `extern int strlen(const char*);`
- ✓ визначення функцій-підстановок `inline char get()`
`{ return *p++; }`
- ✓ оголошення даних `extern int a;`
- ✓ визначення констант `const float pi = 3.141593;`
- ✓ перерахування `enum bool { false, true };`
- ✓ оголошення імен `class Matrix;`
- ✓ команди долучення файлів `#include <signal.h>`
- ✓ макровизначення `#define Case break; case`
- ✓ коментарі `/* Перевірка на кінець файлу */`

Перелік того, що саме слід розміщувати в заголовному файлі, не є вимогою мови C++, це є лише порада розумного використання долучення файлів. З іншого боку, в заголовному файлі ніколи не повинно бути:

- ✓ визначення функцій `char get() { return *p++; }`
- ✓ визначення даних `int a;`
- ✓ визначення складених констант `const tb[i] = { /* ... */ };`

Як вже було зазначено, не можна визначити функцію чи то змінну в заголовному файлі, який використовуватиметься кількома файлами програми. Це призведе до помилок повторних визначень. Проблема виникає й тоді, коли помилково залучають один і той самий заголовний файл двічі.

Отже, мова C++ зручна для створення програм, в яких застосовується кілька файлів водночас. Ефективною організаційною стратегією у цьому відношенні є використання заголовного файлу для визначення типів користувача і прототипів для функцій, які опрацьовують типи користувача. Визначення функцій слід виносити до окремого файлу. В обох файлах – у h-файлі й у crr-файлі визначається і реалізується тип даних користувача, а також те, в який спосіб його може бути використано. Код, який викликає ці функції, слід розміщувати в іншому crr-файлі.

9.3 Директиви препроцесора

Препроцесор – це комп’ютерна програма, яка приймає дані на вході і видає дані, призначені для входження іншої програми, як от компілятор.

У мови програмування С та С++ вбудовано підтримку препроцесора. Рядки в початковому коді, які мають бути опрацьовані препроцесором у вигляді `#define` та `#include`, називаються *препроцесорними директивами*.

Директиви препроцесора є інструкціями, записаними у тексті С-програми, котрі виконуються до трансляції програми. Директиви препроцесора дозволяють змінювати текст програми, наприклад, замінювати певні лексеми в тексті, вставляти текст з іншого файлу, забороняти трансляцію частини тексту тощо. Всі директиви препроцесора розпочинаються зі знаку `#`. Після директив препроцесора крапка з комою не ставиться.

У програмуванні термін «директива» (вказівка) за використанням є схожий до терміну «команда», оскільки так само використовується для опису певних конструкцій мови програмування (тобто вказівок компілятору чи асемблеру стосовно особливостей опрацювання при компіляції).

Узагальнений формат директиви препроцесора є

`# <ім'я_директиви> <лексеми_препроцесора>`

Тут до *лексем препроцесора* належать символні константи, імена файлів, які долучаються, ідентифікатори, знаки операцій, розділові знаки, рядкові константи (рядки) й будь-які символи, відмінні від пробілу.

Залежно від препроцесора перед символом `#` і після нього в директиві можуть бути дозволені пробіли. Пробіли також дозволені перед лексемами препроцесора, поміж ними і після них. За завершення препроцесорної директиви слугує кінець текстового рядка (за наявності символу `'\'`, який позначає перенесення рядка, завершенням препроцесорної директиви буде ознака кінця наступного рядка тексту).

У препроцесорах мови С визначені такі препроцесорні директиви:

`#define` – визначення макроса чи препроцесорного ідентифікатора;

`#include` – долучення тексту з файлу;

`#undef` – скасування визначення макроса чи препроцесорного ідентифікатора;

`#if` – перевірка умови-виразу;

`#ifdef` – перевірка визначеності ідентифікатора;

`#ifndef` – перевірка невизначеності ідентифікатора;

`#else` – початок альтернативної гілки для `#if`;

`#endif` – завершення умовної директиви `#if`;

`#elif` – складена директива `#else/#if`;

`#line` – змінення номера наступного нижче рядка;

`#error` – формування тексту повідомлення про помилку при трансляції;

`#pragma` – дії, передбачені реалізацією;

`#` – порожня директива.

Окрім препроцесорних директив, є три препроцесорні операції, які використовуються разом з командою `#define`:

`defined` – перевірка істинності операнда;

`##` – конкатенація препроцесорних лексем;

`#` – перетворення операнда на рядок символів.

Директива `#define` має кілька модифікацій. Вони допускають визначення макросів та препроцесорних ідентифікаторів, кожному з яких ставиться у відповідність певна послідовність символів. У подальшому тексті програми препроцесорні ідентифікатори замінюються на раніш визначені послідовності символів. Так відбувається, наприклад, при визначенні констант за допомогою директиви `#define`. Директива `#define` слугує для заміни констант, які часто використовуються, ключових слів, операторів чи виразів на певні ідентифікатори. Ідентифікатори, які замінюють текстові чи то числові константи, називають *іменуваними константами*. Ідентифікатори, які замінюють фрагменти програм, називають *макрровизначеннями*, причому макровизначення можуть мати аргументи. Директива `#define` має дві синтаксичні форми:

```
#define <ідентифікатор> <текст>
```

```
#define <ідентифікатор> <(список параметрів)> <текст>
```

Ця директива замінює всі подальші входження ідентифікатора на текст. Такий процес називається *макронідстановкою*. Текст може бути яким завгодно фрагментом програми на C, а також може бути відсутнім. В останньому разі всі екземпляри ідентифікатора вилучаються з програми.

Щоб задати константу `n` зі значенням 10 слід написати директиву

```
#define n 10
```

Ця директива є аналогом такої інструкції

```
const int n = 10;
```

Відмінності полягають у тому, що:

- 1) при використанні константи пам'ять під константну змінну виділяється, а при використанні директиви – ні;
- 2) при використанні директиви не виконується перевірка типу значення, що може спричинитися до помилок у програмі;
- 3) тип значення `n` при використанні директиви визначається автоматично, тобто, якщо у програмі є суттєво, щоб `n` мала тип `unsigned short`, зробити це за допомогою директиви неможливо.

Отже, використання директиви `#define` у C++ без особливої на те потреби є небажаним.

Директива `#include`, яку було розглянуто вище, долучає до тексту програми вміст зазначеного файлу. Ця директива широко використовується для долучення до програм так званих заголовних файлів. Існують три форми цієї директиви:

```
#include <ім'я_заголовного_файлу>
```

```
#include "ім'я_заголовного_файлу"
```

```
#include ідентифікатор_макроса
```

Відмінність між першими двома формами директиви полягає у методі пошуку препроцесором файлу, що долучається. Якщо ім'я файлу записано у кутових дужках, то послідовність пошуку препроцесором заданого файлу у каталогах визначається заданими каталогами включення (include directories). Якщо ім'я файлу записано у лапках, препроцесор шукає файл, переглядаючи каталоги у такій послідовності:

- ✓ каталог того файлу, який містить директиву `#include`;
- ✓ каталоги файлів, які долучили цей файл директивою `#include`;
- ✓ поточний каталог;
- ✓ каталоги, вказані опцією компілятора `/I`;
- ✓ каталоги, задані змінною оточення `INCLUDE`.

Якщо ім'я файлу зазначено зі шляхом, то препроцесор ніде більше цей файл не буде шукати.

Третя форма директиви `#include` припускає наявність макроса, який визначає файл, що долучається. Ідентифікатор макроса не повинен починатися із символів «<» та «"».

Наприклад, для долучення файлу `stdlib.h`, який шукається у стандартному каталозі, директива має вигляд:

```
#include <stdlib.h>
```

Наведемо приклад директиви долучення файлу `stdafx.h`, який шукається передусім у каталозі, де розташований файл, який містить цю директиву:

```
#include "stdafx.h"
```

Покажемо директиви, які долучають файл `C:\Test\My.h`, що шукатиметься лише у каталозі `C:\Test`:

```
#define myincl "C:\Test\My.h"  
#include myincl
```

Директива `#undef` використовується для відмінення чинності поточного визначення директиви `#define` для зазначеного ідентифікатора. Не є помилкою використання директиви `#undef` для ідентифікатора, який не було визначено директивою `#define`. Синтаксис цієї директиви є таким:

```
#undef <ідентифікатор>
```

Як було зазначено вище, заголовний файл також може містити директиви `#include`. Тому іноді складно зрозуміти, які саме заголовні файли долучено до програмного коду, і деякі заголовні файли може бути долучено декілька разів. Уникати цього дозволяють умовні директиви препроцесора.

Розглянемо приклад:

```
#ifndef BIBL_H  
#define BIBL_H  
// Вміст файлу bibl.h  
#endif
```

Умовна директива `#ifndef` перевіряє, чи не було значення `BIBL_H` визначено раніше (`BIBL_H` – це константа препроцесора; такі константи узвичаєно писати великими літерами). Препроцесор опрацьовує наступні рядки аж до дирек-

тиви `#endif`. Інакше він пропускає рядки від `#ifndef` до `#endif`. Директива `#define` `BIBL_H` визначає константу препроцесора `BIBL_H`. Розмістивши цю директиву безпосередньо після директиви `#ifndef`, можна гарантувати, що змістовну частину заголовного файлу `bibl.h` буде включено до початкового тексту лише один раз, скільки б разів не долучався до тексту сам цей файл.

Директива `#if` та її модифікації `#ifdef`, `#ifndef` разом з директивами `#else`, `#endif`, `#elif` дозволяють організувати умовне опрацювання тексту програми. При використуванні цих засобів компілюється не весь текст, а лише ті його частини, які вибираються за допомогою вищенаведених директив. Головна ідея полягає у тому, що, якщо вираз, який розміщено після директив `#if`, `#ifdef`, `#ifndef` виявиться істинним, то буде скомпільовано код, розташований поміж однією з цих трьох директив та директивою `#endif`, інакше цей код буде опущений. Директива `#endif` використовується для позначення завершення блока `#if`. Директиву `#else` можна використовувати з кожною із наведених вище директив для надання альтернативного варіанта компіляції.

Загальна форма запису директиви `#if`:

`#if <константний_вираз>`

Якщо *константний вираз* є істинним, буде скомпільовано код, розташований безпосередньо за цією директивою.

Директива `#line` дозволяє зазначати ім'я файлу і бажаний початковий номер рядка. Базова форма запису цієї команди є такою:

`#line <номер> * <ім'я_файлу> *`

Тут *номер* – це певне додатне число, а *ім'я файлу* – допустимий ідентифікатор файлу. Значення параметру *номер* стає номером поточного початкового рядка, а значення параметру *ім'я файлу* – ім'ям початкового файлу. Параметр *ім'я файлу* є необов'язковий.

Директива `#line` переважно використовується з метою налагодження програм і в спеціальних застосунках.

Директива `#error` дозволяє задавати текст діагностичного повідомлення, яке виводиться при виявленні помилок. За наявності цієї директиви відображується задане повідомлення і номер рядка.

Дія директиви `#pragma` залежить від конкретної реалізації компілятора. Директива дозволяє видавати компіляторові різні інструкції.

Директива `#` є порожньою директивою і завжди ігнорується.

9.4 Область дії та простір імен

Реальний програмний проект розробляється як кілька вихідних файлів, які компілюються окремо один від одного, а потім поєднуються редактором зв'язків. Як було зазначено у підрозд. 8.1, змінні, оголошені всередині тіла функції, називають **локальними**. Такі змінні діють лише всередині своєї функції. Функції з різних файлів можуть використовувати глобально доступні зовнішні змінні. Самі функції зазвичай є зовнішніми і доступними з будь-яких файлів. Кожна змінна характеризується областю дії, областю видимості і три-

валістю життя. Схеми розподілу пам'яті C++ визначають, як довго змінні лишаються в пам'яті, та які частини програми мають до них доступ.

Область видимості змінної – це частина тексту програми, в якій може бути використано змінну. Змінна вважається за видимою, якщо відомі її тип та ім'я. Змінна може бути видимою у межах блока, файлу чи всієї програми. Якщо змінна оголошена у файлі на зовнішньому рівні, вона є видима від оголошення і до кінця цього файлу. Змінну можна оголосити глобально видимою за допомогою відповідних специфікаторів у вихідних файлах програми. Якщо змінна оголошена у блоці, вона є видимою всередині цього блока і блоків, вкладених у нього.

Нагадаємо, що блоком у програмі є складений оператор чи послідовність операторів, обмежених операторними дужками {}. При цьому з точки зору синтаксису блок вважається єдиним оператором. Окрім того, блоки можуть містити складені оператори, але не можуть містити визначення функцій, тобто всередині функції не можна визначити іншу функцію.

Тривалість життя змінної – це інтервал виконання програми, під час якого змінна існує в пам'яті. Змінна з глобальним часом існування має відведену для неї пам'ять упродовж всього часу виконання програми, а змінні з локальним часом життя – лише під час виконання відповідного блока програми. Функції у програмі на C++ мають глобальний час життя.

Як відомо, синтаксис оголошення змінної у програмі є такий:

```
[<клас пам'яті>] [<тип>] <ім'я> = [<ініціалізатор>];
```

Існують такі специфікатори класів пам'яті:

1) auto ;	3) extern ;	5) volatile ;
2) register ;	4) static ;	6) mutable .

Специфікатор **auto** (так звана автоматична змінна) використовується для оголошування лише локальних змінних, які є видимі лише в тому блоці, в якому вони оголошені. Цей специфікатор призначається за промовчанням, а тому його можна явно не зазначати:

```
auto int X=10;           // Те саме, що int X=10;
```

Специфікатор **register** вказує компілятору по можливості розмістити змінну в регістрах процесора. Оскільки кількість регістрів процесора є обмежена, кількість таких змінних має бути невеликою. За невдалої спроби такого розміщення змінна буде вести себе подібно до локальної змінної типу **auto**. Розміщення змінних у регістрах оптимізує програмний код за швидкістю, позаяк процесор оперує зі змінними, розміщеними у регістрах, набагато швидше, ніж з пам'яттю. Регістрова пам'ять, якщо вона є, може бути виділена лише для змінної типу **int** і вказівників:

```
register int Y;
```

Специфікатори **auto** та **register** застосовуються лише для локальних змінних. Спроба застосування цих специфікаторів для оголошення глобальних змінних призведе до помилки.

Специфікатор **extern** (зовнішній) використовують для посилання на змінну, оголошену в іншому файлі (див. підрозд. 9.1). Якщо визначення зовнішньої

змінної розміщено у тому ж самому файлі, але пізніше, ніж її використання, слід теж застосовувати оголошення із специфікатором **extern**.

Специфікатор **static** (статичний) застосовують для оголошення статичних змінних, які зберігають власні значення від одного входження до функції (блока) до наступного входження до цієї функції (блока) протягом усього часу виконання програми. Статичну змінну не можна оголосити в іншому файлі як зовнішню. Змінні зі специфікатором **static** було розглянуто у підрозділах 8.2 та 9.1.

Специфікатор **volatile** (нестабільний) зазначає, що значення змінної (комірки пам'яті) може бути змінено апаратними засобами чи іншою системною програмою протягом виконання програми.

Специфікатор **mutable** (змінюваний) використовується винятково для класів і зазначає, що значення елемента даних завжди є доступне для модифікації, навіть, якщо цей елемент належить до константного об'єкта.

Правила видимості змінних і функцій:

1) змінні, визначені на зовнішньому рівні, є видимі від точки оголошення до кінця файлу. Можна оголосити змінні видимими і в інших вихідних файлах за допомогою специфікатора **extern**. Однак, якщо змінна визначена на зовнішньому рівні зі специфікатором **static**, вона є видима лише в тому файлі, в якому вона визначена;

2) змінна, оголошена чи визначена всередині блока, є видима від точки оголошення до кінця блока;

3) змінні з зовнішніх блоків, включаючи змінні, оголошені на зовнішньому рівні, є видимі у внутрішніх блоках. Якщо змінна, оголошена всередині блока, має те ж саме ім'я, що і змінна, оголошена у зовнішньому блоці, то визначення змінної в блоці змінює зовнішню змінну;

4) функції з класом пам'яті **static** є видимі лише у тому файлі, в якому вони визначені. Решта функцій є глобально видимими, однак для їхнього використання необхідно оголошувати прототипи. Мітка у тілі функції є видима у цьому тілі і тільки у ньому. Імена формальних параметрів функції видимі від точки оголошення до кінця тіла функції.

Правила визначення тривалості життя змінної:

1) змінна, оголошена ззовні блоків програми, має глобальну тривалість життя;

2) змінна, оголошена всередині блока, має локальну тривалість життя, якщо вона оголошена без специфікатора **static**.

Області дії

Кожен програмний об'єкт має область дії, яка визначається видом і місцем його оголошення. Існують такі категорії області дії: блок, файл, функція, прототип функції, клас і поіменована область.

Блок. Ідентифікатори, оголошені всередині блока, є локальними. Область дії ідентифікатора розпочинається з точки визначення і поширюється до кінця блока, видимість ідентифікатора – в межах блока і його внутрішніх блоків, тривалість життя ідентифікатора – до виходу з блока.

Файл. Ідентифікатори, оголошені за межами будь-якого блока, функції, класу чи простору імен, мають глобальну видимість і постійну тривалість життя, і можуть використовуватися з моменту їх визначення у файлі.

Функція. Існує лише один тип ідентифікаторів – мітки операторів, – для яких областю видимості є все тіло функції (незалежно від того, в якому блоці оголошено мітку). У межах однієї функції всі мітки мають бути різними, але імена міток у різних функціях *можуть збігатися*.

Прототип функції. Ідентифікатори, які зазначено у переліку параметрів прототипу (заголовка, оголошення) функції, мають область дії лише прототип функції.

Клас. Елементи структур, об'єднань і класів (за винятком статичних елементів) є видимими лише у межах класу. Вони утворюються при створенні змінної зазначеного типу і руйнуються при її знищенні.

Іменована область. C++ дозволяє явно задавати область визначення імен як частину глобальної області за допомогою оператора `namespace`.

Область видимості співпадає з областю дії, за винятком ситуації, коли у вкладеному блоці оголошено змінну з таким самим ім'ям. У цьому разі зовнішня змінна у вкладеному блоці є невидима, хоча цей блок і входить в її область дії. До цієї змінної, якщо вона є глобальна, можна звертатися, використовуючи операцію визначення діапазону доступу (`::`).

Простори імен

У кожній області дії розрізняють так звані простори імен (`namespace`). Простори імен надають можливість створювати іменовані області пам'яті, в яких можна оголошувати ідентифікатори. Вони призначені для зниження вірогідності конфліктів імен, коли імена з одного простору пам'яті не конфліктують з тими самими іменами, оголошеними в інших просторах пам'яті. Це особливо важливо для великих програм, які використовують програмні коди, розроблені різними постачальниками програмних продуктів. Ідентифікатори у просторах імен можуть стати доступними при застосуванні операції доступу (`::`) або директиви `using`.

У різних просторах імена можуть збігатися, наприклад:

```
struct Node
{ int Node; int i;
} Node;
```

Тут протиріч немає, оскільки імена типу, змінної та елементи структури належать до різних просторів.

У C++ визначено чотири класи ідентифікаторів, у межах кожного з них імена мають бути унікальними:

✓ до першого класу належать імена змінних, функцій, типів користувача (визначених за допомогою `typedef`) і перераховних констант у межах однієї області видимості. Всі вони, окрім імен функцій, можуть бути перевизначені у вкладених блоках;

✓ другий клас імен утворюють імена типів перерахувань, структур, класів й об'єднань. Кожне ім'я має відрізнятися від імен інших типів у тій самій області видимості;

✓ третій клас становлять елементи структури, класу й об'єднання. Ім'я елемента має бути унікальним усередині кожної структури, але може збігатися з іменами елементів інших структур;

✓ мітки операторів утворюють окремий клас імен.

Іменовані області слугують для логічного групування оголошень та обмеження доступу до них. Чим більше є програма, тим більш актуальним є використання іменованих областей. Найпростішим прикладом такого використання є відокремлення коду, написаного однією людиною, від коду, написаного іншою людиною. За використання єдиної глобальної області видимості формувати програму з окремих частин надто складно через можливість збігу й конфлікту імен.

Оголошення іменованої області (її називають також простором імен) має вигляд

```
namespace [<ім'я області>] { <оголошення> }
```

Іменовані області можуть оголошуватись неодноразово, всі наступні оголошення розглядаються як розширення попередніх. Отже, іменована область може поновлюватися і змінюватися за рамками одного файлу.

Якщо ім'я області не є задане, компілятор визначає його самостійно за допомогою унікального ідентифікатора, який є різним для кожного модуля. Оголошення об'єкта в неіменованій області є рівнозначне до його оголошення як глобального із специфікатором **static**. Розміщувати змінні в такій області корисно для того, щоб зберегти локальність коду. Не можна отримати доступ з одного файлу до елемента неіменованої області іншого.

Приклад іменованої області:

```
namespace demo
{ int i=1, k=0;
  void func1(int);
  void func2(int){/* ... */}
}
namespace demo
{ // int i=2; Неправильно – подвійне визначення
  void func1(double);    // Перевантаження
  void func2(int);       // Правильно (повторне оголошення)
}
```

В оголошенні іменованої області можуть бути присутні як оголошення, так і визначення. Логічно розміщувати у ній лише оголошення, а визначати пізніше за допомогою імені області й операції доступу до області видимості (::), наприклад:

```
void demo :: func1(int) {/* ... */}
```

Це застосовується для розділення інтерфейсу й реалізації. У такий спосіб не можна оголосити новий елемент простору імен.

Об'єкти, які було оголошено всередині області, є видимими з моменту оголошення. До них можна явно звертатися за допомогою імені області й операції доступу до області видимості `::`, наприклад:

```
demo::i=100; demo::func2(10);
```

Якщо ім'я часто використовується зовні свого простору, можна оголосити його доступним за допомогою оператора `using`:

```
using demo::i;
```

Після цього можна використовувати ім'я без явного зазначання області.

Якщо є потреба зробити доступними всі імена з певної області, використовується оператор `using namespace`:

```
using namespace demo;
```

Оператори `using` та `using namespace` можна використовувати всередині іменованої області, щоб зробити в ній доступними оголошення з іншої області:

```
namespace Department
{ using demo::i;
  . . .
}
```

Імена, оголошені в іменованій області явно чи за допомогою оператора `using` мають пріоритет щодо імен, оголошених за допомогою оператора `using namespace` (це має значення при долученні кількох іменованих областей, які містять збіжні імена).

Короткі імена просторів імен можуть увійти у конфлікт один з одним, а довгі є непрактичні при написанні реального коду, тому дозволяється вводити синоніми імен:

```
namespace DIT = Department_of_Informational_Technologies;
```

Об'єкти стандартної бібліотеки визначено у просторі імен `std`.

Механізм просторів імен разом з директивою `#include` забезпечує потрібну при написанні великих програм гнучкість завдяки логічному групуванню пов'язаних величин разом з обмеженням доступу.

Здебільшого у кожному завершеному фрагменті програми можна виокремити інтерфейсну частину (наприклад, заголовки функцій, опис типів), потрібну для використання цього фрагмента, і частину реалізації, тобто допоміжні змінні, функції та інші засоби, доступ до яких зовні не потрібний. Простори імен дозволяють приховати деталі реалізації і в такий спосіб спростити структуру програми та зменшити кількість потенційних помилок. Добре помірковане розбиття програми на модулі, чітка специфікація інтерфейсів й обмеження доступу дозволяють організувати ефективну роботу над проектом групи програмістів.

9.5 Бібліотеки функцій

Бібліотеками називають набори функцій (підпрограм) і/або об'єктів, переважно орієнтованих на розв'язання близьких за тематикою завдань. Бібліотеки дозволяють програмісту при написанні нових програм максимально використовувати

ти свої попередні напрацювання з вирішення подібних завдань, долучивши відповідну бібліотеку. Тобто, замість того щоб щоразу реалізовувати одні і ті самі функції для забезпечення тієї чи іншої функціональності у кожному створюваному проекті, їх можна створити один раз і потім викликати з проектів.

З точки зору організації та використання бібліотеки бувають статичними і динамічними.

Статичні бібліотеки (static library) – це набір функцій, але не у формі коду, а в уже скомпільованому вигляді. Файли статичних бібліотек C++ мають розширення `lib`. При зв'язуванні зі статичною бібліотекою програма долучає всі використовувані функції, що збільшує розмір програми, але робить її більш автономною.

Динамічні бібліотеки (dynamic link library – `dll`) завантажуються операційною системою «на вимогу» запущеної програми вже в ході її виконання. Якщо необхідна бібліотека вже була завантажена у пам'ять, то повторне завантаження не виконуватиметься. При цьому один і той самий набір функцій чи об'єктів бібліотеки може бути використаний водночас кількома працюючими програмами, що дозволяє ефективно використовувати ресурси оперативної пам'яті. Динамічні бібліотеки у Windows зазвичай мають розширення `dll`.

Бібліотеки динамічного компонування, так само, як і статичні бібліотеки, є двійковими файлами, що містять функції, а в деяких випадках дані і ресурси, які можуть бути використані у проекті (або в іншій бібліотеці динамічного компонування).

Існує кілька причин використання бібліотек динамічного компонування, основними з яких є:

- ✓ модульність застосунку, що полегшує внесення змінень і повторне використання його складових частин;
- ✓ поділ бібліотеки між декількома одночасно відкритими програмами.

Перш ніж деякий процес зможе використовувати бібліотеку динамічного компонування, необхідно спочатку завантажити в оперативну пам'ять файл образу бібліотеки динамічного компонування і включити його в адресний простір цього процесу. Це може бути зроблено одним з двох способів: неявним компонуванням у процесі завантаження або явним компонуванням у процесі виконання завдання.

Розглянемо на прикладах процесу створення статичної і динамічної бібліотек у Visual Studio 2017.

Приклад 9.1. Створити статичну бібліотеку з функціями, які реалізовуватимуть видалення та вставлення певної частини рядка. Також створити консольний проект, який використовуватиме функції бібліотеки для введення рядка.

Розв'язок. У бібліотеці буде дві функції, кожна з яких матиме по три аргументи. Функція видалення певної частини рядка `DelChars()` матиме такі аргументи:

- ✓ рядок, з якого треба видалити частину символів;
- ✓ позиція (індекс) символу, з якого треба починати видалення;
- ✓ кількість символів, що мають бути видалені.

Аргументи функції вставлення `Insert()` будуть такі:

- ✓ рядок, в який слід вставити підрядок (слово);
- ✓ підрядок (слово), яке вставляється;
- ✓ позиція, на яку вставляється підрядок (слово).

Алгоритм створення статичної бібліотеки складатиметься з таких етапів:

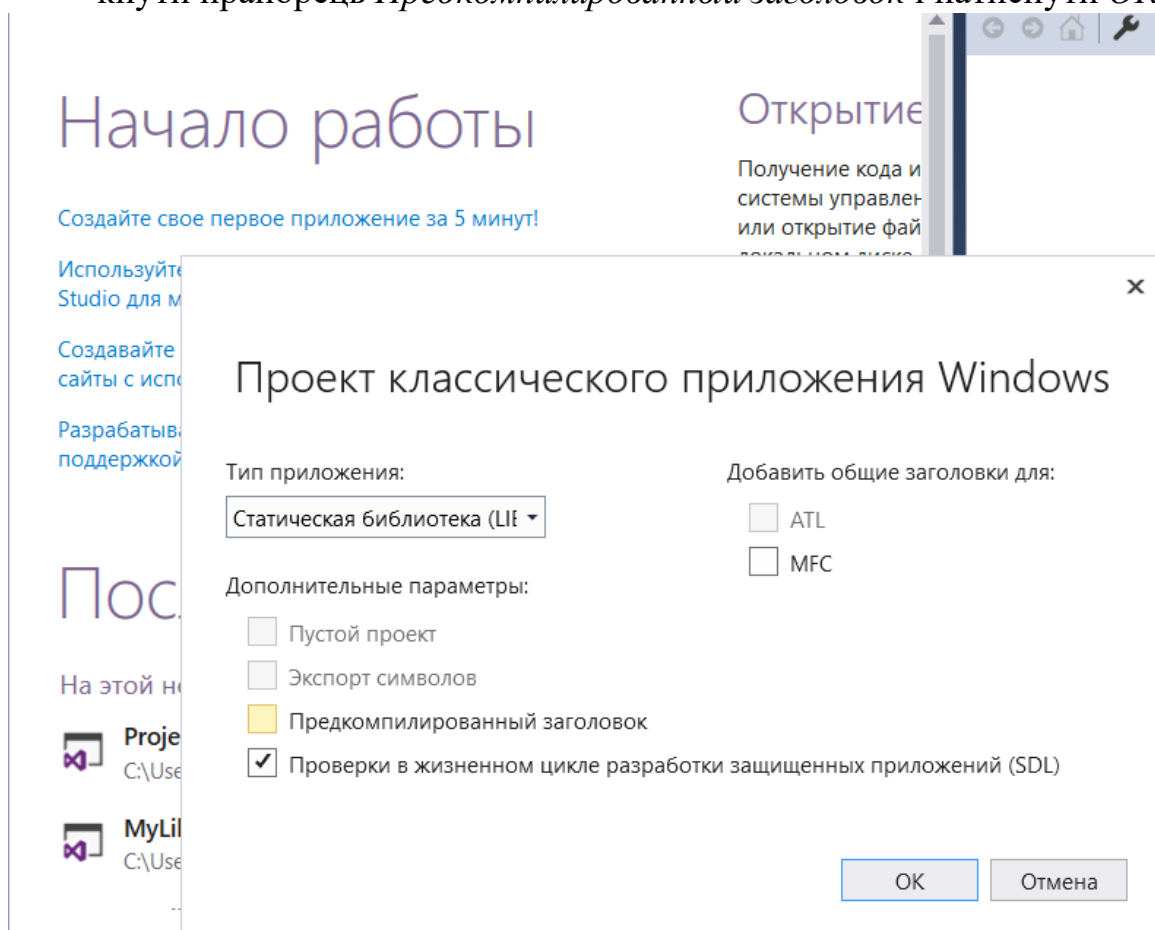
1. створення статичної бібліотеки у Visual Studio;
2. додавання класу до статичної бібліотеки;
3. створення та налаштування консольного проекту в Visual Studio, який використовуватиме створену бібліотеку.

Розглянемо докладно кожен з цих етапів.

1. Створення статичної бібліотеки у Visual Studio

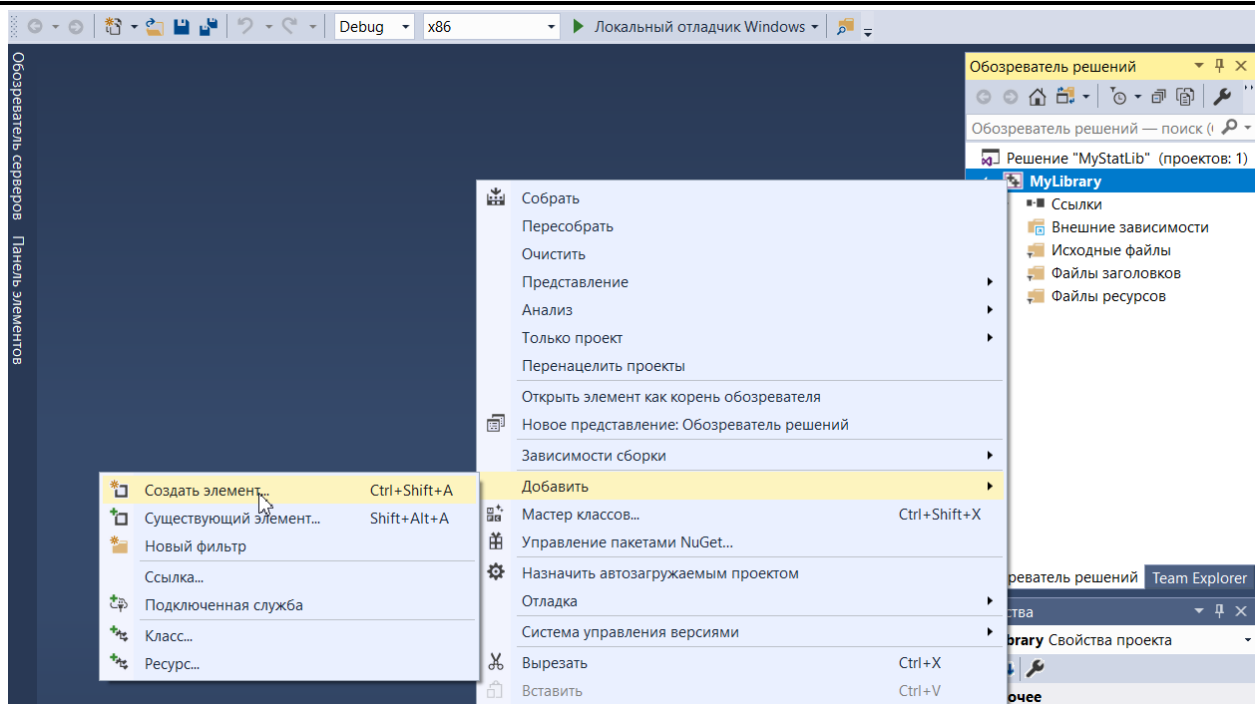
1.1. Спочатку треба при створенні нового проекту вибрати *Мастер создания классических приложений Windows (Windows Desktop Wizard)* і ввести назву проекту, наприклад **MyLibrary**, і назву рішення, наприклад **MyStatLib**.

1.2. У вікні майстра вибрати тип проекту *Статическая библиотека (.lib)*, вимкнути прапорець *Предкомпилированный заголовок* і натиснути **OK**.



2. Додавання класу в статичну бібліотеку

2.1. Для створення файлу заголовка для нового класу треба в оглядачі розв'язків відкрити контекстне меню проекту **MyLibrary**, а потім вибрати команду *Добавить – Создать элемент*.



2.2.У діалоговому вікні додавання нового елемента ліворуч у розділі *Visual C++* треба вибрати *Код*. У центральній області вибрати *Заголовочный файл (.h)*, вказати ім'я для файлу заголовка, наприклад **MyLibrary.h**, і натиснути кнопку *Добавить*. З'явиться порожній файл заголовка.

2.3.Створити простір імен **StringFuncs** і клас **MyStrFuncs**:

```
namespace StringFuncs
{
    class MyStrFuncs
    {
    public:
        static void DelChars(char* s, int p, int l);
        static char* Insert(char* str, const char* spos, int pos);

    };
}
```

Модифікатор `static` використовується тут для оголошення статичного члена класу, що належить власне типу, а не конкретному об'єкту. Доступ до цих методів буде здійснюватися із використанням імені класу та оператора доступу (`::`).

2.4.При створенні вихідного файлу для нового класу слід в оглядачі розв'язків відкрити контекстне меню для проекту **MyLibrary** і вибрати *Добавить – Создать элемент*. У діалоговому вікні додавання нового елемента у лівій області розділу *Visual C++* вибрати *Код*, після чого у центральній області вибрати *Файл ++ (.cpp)*. Вказати ім'я для файлу заголовка, наприклад **MyLibrary.cpp**, і натиснути кнопку *Добавить*. Після цього з'явиться вікно порожнього вихідного файлу.

2.5.У цьому файлі слід ввести код реалізації функцій MyStrFuncs:

```
#include "MyLibrary.h"
using namespace std;

namespace StringFuncs
{
    void MyStrFuncs::DelChars(char* s, int p, int l)
    {
        char* dst;
        for (dst = s + p, s = dst + l; *dst++ = *s++; );
    }

    char* MyStrFuncs::Insert(char* str, const char* spos, int pos)
    {
        char* tmp = str;
        const char* ps;
        int len1, len2, cnt, i;
        for (len1 = 0, ps = spos; *ps; *ps++, len1++);
        for (len2 = 0, ps = str; *ps; *ps++, len2++);
        cnt = len1;
        while (cnt--)
        {
            for (i = len2 + len1; i >= pos; i--)
                *((str)+i + 1) = *((str)+i);
        }
        str += pos;
        while (*spos)
            *str++ = *spos++;
        return tmp;
    }
}
```

2.6.Скомпілювати бібліотеку командою *Сборка – Собрать решение*.

3. Створення та налаштування консольного проекту в Visual Studio, який використовуватиме створену бібліотеку

3.1.При створенні нового проекту вибрати *Мастер классических приложений Windows*.

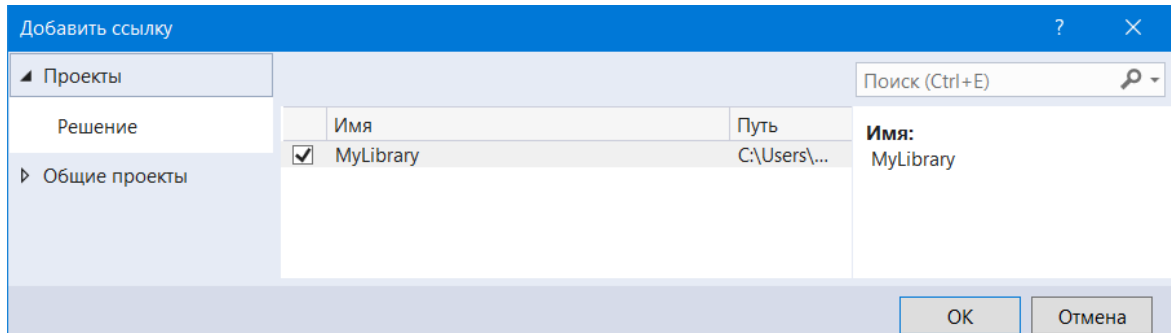
3.2.Вказати ім'я проекту, наприклад Project1, у полі *Имя*. Зі списку поруч з полем *Решение* вибрати *Добавить в решение*. В результаті новий проект буде додано до рішення, що містить статичну бібліотеку. Натиснути кнопку *ОК*.

3.3.У вікні майстра вибрати тип проекту *Консольное приложение*. Вимкнути прапорець *Предкомпилированный заголовок* і натиснути *ОК*.

3.4.Для використання функцій зі статичної бібліотеки у проекті необхідно послатися на цю бібліотеку. Для цього треба в оглядачі розв'язків для

проекту Project1 натиснути правою кнопкою миші і вибрати *Ссылки – Добавить ссылку*.

- 3.5. У діалоговому вікні додавання посилання перераховані бібліотеки, на які можна сформулювати посилання. На вкладці *Проекты* буде наведено усі проекти поточного рішення і включені у них бібліотеки, якщо вони є. Слід увімкнути прапорець *MyLibrary*, а потім натиснути кнопку *OK*.



- 3.6. Для створення посилання на файл заголовка *MyLibrary.h* необхідно змінити шлях до папки з цим файлом. Для цього у діалоговому вікні властивостей проекту Project1 (*Проект – Свойства: Project1*) треба розгорнути вузли *Свойства конфигурации – C / C++ – Общие*. У полі *Дополнительные каталоги включаемых файлов* слід вказати шлях до папки *MyLibrary* або відшукати цю папку. Для цього треба відкрити список значень властивості і вибрати *Изменить*. У текстовому полі діалогового вікна *Дополнительные каталоги включаемых файлов* клацнути на порожній рядок і натиснути кнопку з трьома крапками (...) наприкінці рядка. У діалоговому вікні *Выбор каталога* вибрати папку *MyLibrary* і натиснути кнопку *Выбор папки*. Натиснути двічі *OK*.

- 3.7. Тепер клас *MyStrFuncs* можна використовувати в проекті. Для цього слід замінити вміст файлу *Project1.cpp* на такий код:

```
#include "MyLibrary.h"
#include <iostream>

using namespace std;
int main()
{
    char s[100], str[100]; int pos;
    setlocale(LC_ALL, ".1251"); puts("Введіть рядок: ");
    setlocale(LC_ALL, ".OCP"); gets_s(s, 100);
    StringFuncs::MyStrFuncs::DelChars(s, 4, 7);
    puts(s);
    setlocale(LC_ALL, ".1251"); puts("Введіть рядок для вставлення: ");
    setlocale(LC_ALL, ".OCP"); gets_s(str, 100);
    setlocale(LC_ALL, ".1251");
    puts("Введіть позицію для вставлення (ціле число): ");
    cin >> pos;
```

```

setlocale(LC_ALL, ".OCP");
puts(StringFuncs::MyStrFuncs::Insert(s, str, pos));
system("pause");
return 0;
}

```

Звертаємо увагу на те, що для виклику створених функцій DelChars і Insert треба спочатку вказати назву простору імен (StringFuncs), а потім назву класу (MyStrFuncs) та використати оператор доступу (::).

3.8.Щоб призначити консольний проект Project1 таким, що завантажується першим, слід натиснути на проекті правою кнопкою миші і вибрати *Назначить автозагружаемым проектом*.

3.9.Скомпілювати проект командою *Сборка – Собрать решение* і запустити на виконання.

Результати виконання програми:

Введіть рядок:

Один два три чотири

Один чотири

Введіть рядок для вставлення:

шість

Введіть позицію для вставлення (ціле число):

5

Один шість чотири

Приклад 9.2. Створити динамічну бібліотеку з функціями, які реалізовуватимуть такі завдання з одно- та двовимірними масивами:

1) обчислення елементів матриці a розмірністю 8×8 за формулою

$$a_{i,j} = \begin{cases} 3 \sqrt{\left(\frac{i+j+2}{7i+1} + j \right)^4} - 3.7 * (i+j), & \text{за } \sin(i+j) > 0 \\ \pi \sin^3(i-j), & \text{за } \sin(i+j) \leq 0 \end{cases}$$

2) обчислення вектора (одновимірного масиву), елементи якого є сумами елементів відповідних рядків матриці a ;

3) пошук елемента матриці a , найбільш наближеного до значення середнього арифметичного елементів матриці a .

Після створення DLL-бібліотеки створити консольний проект, який використовуватиме функції цієї бібліотеки.

Розв'язок. У бібліотеці буде три функції. Алгоритм створення DLL-бібліотеки з цими функціями у Visual Studio 2017 складатиметься з таких етапів:

1. створення проекту бібліотеки DLL;
2. створення заголовного файлу (.h) з оголошенням констант і заголовків функцій;
3. реалізація функцій (оголошених у заголовному файлі) у .cpp файлі;
4. створення консольного проекту, який використовуватиме створену DLL-бібліотеку.

Розглянемо докладно кожен з цих етапів.

1. Створення проекту бібліотеки DLL у Visual Studio.

- 1.1. При створенні нового проекту вибрати *Мастер создания классических приложений Windows (Windows Desktop Wizard)* і ввести назву, наприклад, **MyLib**.
- 1.2. У вікні майстра вибрати тип проекту *Библиотека динамической компоновки (.dll)*.
- 1.3. У властивостях проекту (меню *Проект – Свойства*) у розділі *C/C++ – Препроцессор* перевірити визначення препроцесора: **MYLIB_EXPORTS** має бути написано великими літерами. Якщо це не так (написано **MyLib_EXPORTS**), треба виконати такі дії:
 - ✓ вгорі вікна властивостей вибрати конфігурацію *Все конфигурации*;
 - ✓ у розділі *C/C++ – Препроцессор – Определения препроцессора* клацнути на стрілку праворуч і натиснути *Изменить*;
 - ✓ у верхній панелі додати **MYLIB_EXPORTS**;
 - ✓ двічі клацнути *ОК*.

2. Створення заголовного файлу (.h) з оголошенням констант та заголовків функцій.

- 2.1. Для створення заголовного файлу треба в меню виконати команду *Проект – Добавить новый элемент* і вибрати *Заголовочный файл (.h)*. Задати **MathLib.h** як ім'я файлу.
- 2.2. Ввести такий програмний код з оголошеннями у заголовний файл **MathLib.h**:

```
#pragma once
#define k 8    // Оголошення константи
// Оголошення типу matr (матриця) і vekt (вектор)
typedef double matr[k][k], vekt[k];
// Оголошення прототипів функцій
__declspec(dllexport) void elem_Matr(matr c);
__declspec(dllexport) void elem_Vect(matr c, vekt v);
__declspec(dllexport) double G(matr c);
```

Тут кожна з трьох функцій, які будуть експортуватися, оголошується з ключовим словом: **__declspec(dllexport)**.

3. Реалізація функцій у .cpp файлі.

- 3.1. Перейти на вкладку **MyLib.cpp**, якщо вона вже відкрита. Якщо ні, то відкрити її в оглядачі розв'язків у папці *Исходные файлы*.
- 3.2. Ввести такий програмний код у файл реалізації **MyLib.cpp**:

```
...
#define _USE_MATH_DEFINES
#include <math.h>    // Долучення математичної бібліотеки
#include "MyLib.h"
```

```
// Визначення функції розрахунку елементів матриці
void elem_Matr(matr c)
{
    for (int i = 0; i < k; i++)
        for (int j = 0; j < k; j++)
            if (sin(i + j) > 0)
                c[i][j] = pow((i+j+2.0)/(7*i+1)+j, 4.0/3) - 3.7*(i+j);
            else
                c[i][j] = M_PI * pow(sin(i - j), 3);
}
// Визначення функції розрахунку елементів вектора
void elem_Vect(matr c, vekt v)
{
    for (int i = 0; i < k; i++)
    {
        v[i] = 0;
        for (int j = 0; j < k; j++) v[i] += c[i][j];
    }
}
// Пошук найближчого елемента до середнього арифметичного
double G(matr c)
{
    int ni = 0, nj = 0, i, j;
    double sr = 0;
    for (i = 0; i < k; i++)
        for (j = 0; j < k; j++)
            sr += c[i][j] / pow(k, 2);
    for (i = 0; i < k; i++)
        for (j = 0; j < k; j++)
            if (fabs(sr - c[i][j]) < fabs(sr - c[ni][nj]))
            {
                ni = i;    nj = j;
            }
    return c[ni][nj];
}
```

3.3.Скомпілювати створену динамічну бібліотеку командою *Сборка – Построить решение*. Якщо помилок не буде виявлено, це означатиме, що бібліотеку успішно створено. Інакше треба буде виправити помилки і знову виконати компіляцію.

4. Створення консольного проекту, який використовуватиме створену DLL-бібліотеку.

4.1.Створити новий консольний проект Windows.

4.2.Відкрити властивості проекту, вибрати конфігурацію *Все конфигурации*, перейти у розділ *C/C++ – Общие*. Клацнути праворуч від пункту *Дополнительные каталоги включаемых файлов*, вибрати *Изменить* і

вказати шлях до папки, в якій зберігається заголовний файл MyLib.h (...\\MyLib\\MyLib). Клацнути *OK*.

4.3. Для використання функцій бібліотеки MyLib.h у створеному консольному проєкті треба ввести такий програмний код:

```
#include "pch.h"
#include <iostream>
#include <iomanip>
#include "MyLibrary.h"      // Долучення власної бібліотеки
using namespace std;

int main()
{
    matr a;   vekt x;
    elem_Matr(a); // Виклик функції обчислення елементів матриці
    cout << "Matrix:\n";
    cout << fixed;
    for (int i = 0; i < k; i++)
    {
        for (int j = 0; j < k; j++)
            cout << setprecision(2) << a[i][j] << setw(8);
        cout << endl;
    }
    elem_Vect(a, x); // Виклик функції обчислення елементів вектора
    cout << "\nVector:\n";
    for (int i = 0; i < k; i++)
        cout << setprecision(2) << x[i] << setw(8);
    cout << endl;
    // Виведення значення скаляра
    cout << "\nG=" << G(a) << endl;
    system("pause");
    return 0;
}
```

Цей код можна скопіювати, але він не може бути приєднаним, позаяк ще не вказано шлях до файлу MyLib.lib.

4.4. Відкрити вікно властивостей проєкту, вибрати розділ *Компоновщик – Ввод – Дополнительные зависимости*. Як і на попередньому кроці вибрати *Изменить* і ввести у верхній панелі назву MyLib.

4.5. У вікні властивостей проєкту перейти до розділу *Компоновщик – Общие – Дополнительные каталоги библиотек*. Як і на попередньому кроці вибрати *Изменить* і вказати у верхній панелі шлях до папки, яка містить файл MyLib.lib (...\\MyLib\\Debug).

Консольний застосунок відтепер успішно компілюється та з'єднується, однак все ще не може бути виконаним.

4.6. У вікні властивостей проєкту перейти до розділу *События сборки – Событие после сборки*, ввести у командний рядок такий текст:
хсору /y /d "..\\..\\MyLib\\\$(IntDir)MyLib.dll" "\$(OutDir)".

4.7. Виконати в меню команду *Сборка – Собрать решение*.

Результати роботи консольного проекту:

Matrix:

```
0.00    2.65    3.50    4.90    1.36    2.77    0.07    14.42
-3.43   -5.68   -7.48   -2.36   -0.01    1.36   -12.19   -12.93
-7.23   -9.63    0.00   -1.87   -2.36   -15.96   -17.05   -18.01
-10.96    2.36    1.87    0.00   -18.67   -20.00   -21.17    1.36
-1.36    0.01    2.36   -20.97   -22.51   -23.87   -2.36   -0.01
-2.77   -1.36   -22.95   -24.73   -26.30    0.00   -1.87   -2.36
-0.07   -24.61   -26.68   -28.47    2.36    1.87    0.00   -33.81
-25.80   -28.32   -30.40   -1.36    0.01    2.36   -36.46   -37.59
```

Vector:

```
29.67  -42.72  -72.11  -65.20  -68.71  -82.34 -109.42 -157.57
```

G=-9.63

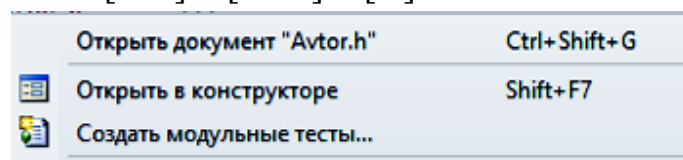
9.6 Створення проектів з декількома формами

Усталено у Visual Studio створюваний проект з формою містить файли для однієї форми та вихідного коду одного модуля. Однак більшість більш-менш серйозних проектів містять декілька форм і модулів. Для створення і долучення нової форми слід виконати команду *Проект / Додати новий елемент / Форма Windows Forms*, ввести ім'я форми та натиснути кнопку *Додати*. Розмістити на щойно створеній формі потрібні елементи та створити відповідний програмний код.

Залучити заголовний файл створеної форми чи модуля можна прописавши директиву `#include`. Для цього треба повернутися до файлу `Form1.h`, піднятися на початок проекту і після команди `#pragma once` вписати команду долучення щойно створеної форми:

```
#include "Avtor.h"
```

Коли у подальшій роботі виникає потреба відкрити файл долученого модуля, можна скористатись командою контекстного меню *Открыть документ* або комбінацією клавіш [Ctrl] + [Shift] + [G].



Приклад 9.3. Створити програмний проект з двома формами в одному програмному проекті, а саме: до проекту з формою долучити форму-заставку з тематичним рисунком, яка автоматично з'являтиметься після запуску програми та через декілька секунд «розчиниться у повітрі».

Розв'язок. Для створення і долучення нової форми-заставки слід виконати команду *Проект / Додати новий елемент / Форма Windows Forms*, ввести ім'я форми – *Zastavka* та натиснути кнопку *Додати*.

Розмістити на формі елементи `timer1` та `pictureBox1`, зображення до якого завантажити за допомогою властивості `Image`.



Подвійним клацанням по таймеру та по формі створити шаблони функцій і вписати у них такий програмний код:

```
private: System::Void Zastavka_Load(System::Object^ sender, System::EventArgs^ e)
{
    this->timer1->Start();
}
private: System::Void timer1_Tick(System::Object^ sender, System::EventArgs^ e)
{
    // кожної 0.1 секунди прозорість форми зменшуватиметься на 2%
    this->Opacity -= 0.02;
    if (this->Opacity == 0) Close();
}
```

Тепер треба повернутись до файлу `Form1.h`, піднятися на початок проекту і долучити модуль щойно створеної форми `Zastavka`:

```
#include "Zastavka.h"
```

Крім того, слід віднайти функцію `Form1_Load()` та дописати до тих операторів, що там є, ще три оператори для запуску форми-заставки:

```
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
    this->Hide();
    Zastavka^ fr2 = gcnew Zastavka();
    fr2->ShowDialog();
}
```

Запустити проект та впевнитись у правильності роботи всіх створених функцій.

Питання та завдання для самоконтролю

- 1) У чому полягають особливості оголошення простої змінної з використанням зарезервованого слова `extern`?
- 2) Чим оголошення `int X` відрізняється від `extern int X`?
- 3) Чим оголошення `static int X` відрізняється від `extern int X`?
- 4) Коли доречно оголошувати змінні як `extern`? А коли як `static`?
- 5) Чим відрізняється оголошення від визначення змінних у багатофайлових проектах?
- 6) Назвати можливі варіанти розміщення прототипів функцій?
- 7) Яке призначення заголовних файлів?
- 8) Що міститься у файлах реалізації?
- 9) За допомогою якої препроцесорної директиви долучають заголовні файли?
- 10) Чим різняться директиви `#include "bibl.h"` та `#include <bibl.h>`?
- 11) Для чого у мові C++ використовують директиви препроцесора?
- 12) Назвати відомі Вам препроцесорні директиви.
- 13) Назвати умовні директиви опрацювання текстів програм.
- 14) Назвати й охарактеризувати категорії області дії програмних об'єктів.
- 15) Розтлумачити відмінності між такими задаваннями констант:

```
#define nom 256  
const int nom = 256;
```
- 16) Що розуміють під поняттям «тривалість життя змінної»?
- 17) Які існують специфікатори класів пам'яті при оголошенні змінних у програмі? Який саме специфікатор призначається за промовчанням?
- 18) У чому схожість та відмінність, якщо такі є, в оголошеннях `int X=10` та `auto int X=10`?
- 19) Чи можна всередині функції визначити іншу функцію?
- 20) Назвати правила видимості змінних і функцій.
- 21) Чи можна оголошувати функції з класом пам'яті `static`? Чому?
- 22) З якою метою оголошуються іменовані області (простори імен)?
- 23) Охарактеризувати використання операції доступу «`::`».
- 24) Що являють собою бібліотеки функцій C++?
- 25) Які бібліотеки можна створювати у Visual C++?
- 26) Чим відрізняються статичні та динамічні бібліотеки?
- 27) Чому по суті бібліотеки динамічного компонування, так само, як і статичні бібліотеки, є двійковими файлами?
- 28) У чому переваги використання DLL-бібліотек перед LIB-бібліотеками?
- 29) З яких етапів складається алгоритм створення статичної бібліотеки?
- 30) З яких етапів складається алгоритм створення динамічної бібліотеки?

РОЗДІЛ 10

ТИПИ КОРИСТУВАЧА

10.1 Перейменовування типів (typedef)

C++ надає можливість моделювання нових типів даних на базі вже наявних типів. Як приклад доцільності застосовування таких типів розглянемо оголошення

```
double A[5][7];
```

Тут оголошується змінна *A* як матриця (двовимірний масив) дійсних чисел розмірності 5 рядків і 7 стовпчиків. Якщо у програмі оголошується кілька аналогічних масивів або якщо програма містить функції, параметрами яких є матриця 5×7 , зручніше є створити окремий тип «матриця» для оголошення змінної *A*, щоб уникнути помилок і покращити читабельність тексту програми. У мові C++ для створювання типів користувача на базі вже наявних типів використовується службове слово **typedef** (від англійської «type definition» – означення типу). Отже оголошення описаного типу *matrica* може мати вигляд:

```
typedef double matrica [5][7];
```

Цей тип є синонімом дійсного двовимірного масиву розмірності 5×7 , і його можна використовувати як звичайний тип для оголошення змінної *A*:

```
matrica A;
```

Це оголошення означає, що змінна *A* має тип *matrica*, тобто є дійсним двовимірним масивом. Після перейменування типу можна використовувати новий тип *matrica* для оголошення всіх дійсних двовимірних масивів 5×7 , наприклад:

```
matrica B, C;           // B і C – матриці 5x7
double sumr(matrica C); // Заголовок функції, параметром якої є матриця 5x7
matrica M[4];           // Масив з 4-х матриць
```

Порівняймо останнє оголошення з еквівалентним до нього, але менш зручним, без перейменування типу:

```
double M[4][5][7];
```

Перейменування типу у загальному вигляді має синтаксис:

```
typedef <тип> <нове_імя> [<розмірність>];
```

Тут квадратні дужки є елементом синтаксису. Розмірність може бути відсутньою.

Наведемо приклади перейменовування типів **typedef**:

```
typedef unsigned char byte; /* Перейменування типу
                             «беззнаковий char» на byte */
typedef char MS[50];        /* Перейменування типу «рядок
                             з 50-ти символів» на MS */
```

Розглянемо ще один приклад оголошення без перейменовування:

```
char s[20];                // Слово s – рядок з 20-ти символів,
char mas[10][20];          // mas – масив з 10 слів
```

Перейменування типу «рядок з 20-ти символів» на тип «слово»:

```
typedef char slovo [20];  
slovo s;           // Оголошення змінної s типу slovo  
slovo mas[10];     // і масиву з 10-ти слів
```

Взагалі `typedef` є просто засобом спрощення запису операторів оголошення змінних.

Доволі часто визначення типів даних використовується разом зі структурами, що надає можливість створювання складних типів даних, які поєднують різнотипні характеристики. Приклади такого використання буде наведено у наступному підрозділі 10.3.

10.2 Об'єднання (union)

Об'єднання – формат даних, який може містити різні типи даних, але лише один тип водночас. Можна сказати, що об'єднання є окремим випадком структури, всі поля якої розташовуються за однією й тією самою адресою. Формат опису об'єднання є такий самий як і у структури, лише замість ключового слова `struct` використовується слово `union`. Але, тоді як структура може містити, скажімо, елементи типу і `int`, і `short`, і `double`, об'єднання може містити чи то `int`, чи `short`, чи `double`:

```
union prim  
{ int x;  
  short y;  
  double z;  
};
```

Обсяг пам'яті, яку займає об'єднання дорівнює найбільшому з розмірів його полів. Об'єднання застосовують для економії пам'яті у тих випадках, коли відомо, що понад одного поля водночас не потрібно. Часто об'єднання використовують як поле структури. При цьому до структури зручно включити додаткове поле, яке визначає, який саме елемент об'єднання використовується даного моменту:

```
struct  
{ int type;  
  union id  
  { long id_num; char id_ch[20]; };  
} price;  
.  
.  
.  
if (price.type == 1) cin >> price.id.id_num;  
else    cin >> price.id.id_ch;
```

Тут структура `price` містить ціле поле `type` та поле `id`, яке може набувати лише одне зі значень чи то цілого числа `id_num`, чи то рядка `id_ch` залежно від значення поля `type`. У наведеному прикладі ім'я об'єднання можна не зазначати, а звертатися безпосередньо до його полів. У цьому разі об'єднання є анонімним, його елементи стають змінними, розташованими за однією адресою.


```

struct
{ int type;
  union
  { long id_num;
    char id_ch[20];
  };
} price;
...
if (price.type == 1) cin >> price.id_num;
else cin >> price.id_ch;

```

10.3 Структури (struct)

Поняття структури

У попередніх розділах для зберігання даних розглянуто прості типи даних: числа, символи, рядки тощо. Але часто на практиці перед програмістом постає завдання запрограмувати той чи інший об'єкт, який характеризується водночас кількома параметрами, наприклад, точка на площині задається парою дійсних чисел (x, y) , а дані про людину можна задавати кількома параметрами: прізвище, ім'я, по батькові – рядки, рік народження – число тощо. Для поєднання кількох параметрів в одному об'єкті в C++ існує спеціальний тип даних, який має назву структура. На відміну від масивів, які також є структурованим типом даних і містять елементи одного типу, структури дозволяють поєднувати дані різних типів.

Структура – це тип даних, який може поєднувати у собі різнотипні елементи. Елементи структури називаються *полями структури* і можуть мати будь-який тип, крім типу тієї самої структури, але можуть бути вказівником на нього.

Опис структури має синтаксис:

```

struct <ім'я_типу_структури>
{ <тип1> <поле1>;
  <тип2> <поле2>;
  ...
  <типN> <полеN>;
} <список_змінних>;

```

Ім'я типу структури задається програмістом виходячи зі змісту даних, які зберігатимуться у структурі. Список змінних наприкінці оголошення структури може бути відсутнім, у цьому разі після фігурної дужки має бути крапка з комою.

Наведемо приклад оголошення структури з ім'ям `sportsman`, яка поєднає в собі інформацію про змагання спортсменів: прізвище і вік спортсмена, країна, кількість його очок, утворюючи відповідні поля:

```

struct sportsman
{ char prizr[15]; char kraina[10]; // Прізвище та країна,
  int vik;         float ochki;   // вік та кількість очок
};

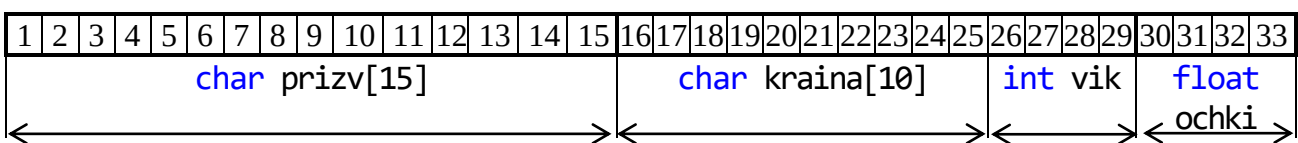
```


Як наслідок цього оголошення створено новий тип `sportsman`. Тепер у програмі цей тип може використовуватись нарівні зі стандартними типами для оголошення змінних. Оголошені змінні типу `sportsman` матимуть чотири поля: два рядки (`prizv` і `kraina`), ціле (`vik`) і дійсне (`ochki`) числа.

```
sportsman Sp;                /* Змінна Sp муну sportsman може зберігати
                               інформацію про одного спортсмена */

sportsman Mas[15];           /* Масив Mas муну sportsman може містити
                               інформацію про 15 спортсменів */
```

При оголошуванні змінної типу структури пам'ять під усі поля структури виділяється послідовно для кожного поля. У наведеному прикладі структури `sportsman` під змінну `Sp` послідовно буде виділено 15, 10, 4, 4 байти – усього 33 байти. Поля змінної `Sp` у пам'яті буде розміщено у такому порядку:



Слід зазначити, що сумарну пам'ять, яка виділяється під структуру, може бути збільшено компілятором на вимогу процесора для так званого вирівнювання адрес. Реальний обсяг пам'яті, яку займає структура, можна визначити за допомогою `sizeof()`.

При оголошуванні структур їхні поля можна ініціалізовувати початковими значеннями. Наприклад, оголошення змінної `Sp` типу `sportsman` й ініціалізація її даними про 20-річного спортсмена з України за прізвищем Бойко, який набрав 75.3 очок, матиме вигляд:

```
sportsman Sp={"Бойко", "Україна", 20, 75.3};
```

Оголошення типу «точка на площині» `POINT` з полями-координатами – `x` та `y` і змінної `spot` цього типу – точки з координатами (20, 40) може бути таким:

```
struct POINT
{ int x, y;
  } spot = { 20, 40 };      // Точка має координати x = 20, y = 40
```

При ініціалізації масивів структури кожен елемент масиву слід записувати у фігурних дужках, наприклад, при створенні двовимірної масиву `comp1` розміром 2×3 типу структури `complex` з полями `real` та `im` ініціалізація початкових значень елементів може мати вигляд:

```
struct complex
{ double real, im;
  } comp1[2][3]={{{1.4}, {5}, {-3.1}},{{2}, {-5.2}, {0}}};
```

Доступ до полів структури

Доступ до полів структури здійснюється за допомогою операції «крапка»:

`<структурна змінна>.<ім'я поля>`

Наприклад, для розглянутої змінної `Sp` типу `sportsman` надання значень її полям може мати вигляд:

```
strcpy(Sp.prizv, "Бойко");
strcpy(Sp.kraina, "Україна");
Sp.vik = 20;
Sp.ochki = 75.3;
```

Можна створювати тип структури за допомогою **typedef**, наприклад:

```
typedef struct
{ String^ priz, ^gruppa;
  int Bal_math, Bal_inform;
} student;           // Оголошення типу «студент»
student W;           // і змінної W цього типу
```

Тут означено тип структури за ім'ям student для зберігання даних про успішність студентів з полями: прізвище, група, екзаменаційні оцінки з математики та інформатики й оголошено змінну W цього типу.

Структури можуть розміщуватись у динамічній пам'яті за допомогою оператора **new** і оголошуватися як вказівники. У такому разі доступ до полів структури здійснюється за допомогою операції вибору – «стрілка» (->, на клавіатурі послідовно натискаються клавіші «мінус» і «більше»):

```
sportsman *p=new sportsman;    // Виділення пам'яті,
p->ochki=95.3;                  // присвоєння очок полю ochki
(*p).vik=23;                    // і віку полю vik
```

Зверніть увагу на те, що якщо p – вказівник на структуру, то (*p) – сама структура і доступ до її полів здійснюється за допомогою крапки.

Якщо змінні типу «структура» оголошуються у програмі лише один раз, то зазвичай їх записують наприкінці оголошення структури і при цьому ім'я структури можна явно не зазначати, наприклад:

```
struct
{ char prizv[15];
  char kraina[10];
  int vik; float ochki;
}
Sp;
```

Тут оголошено змінну Sp як структуру з чотирма полями без створення типу «спортсмен».

Поля структури можуть бути якого завгодно типу, у тому числі й масивом, вказівником чи структурою, окрім типу тієї ж самої структури (але можуть бути вказівником на неї). За приклад створимо дві структури для опрацювання інформації про співробітників у відділі кадрів: прізвище, ім'я, по батькові співробітника, домашня адреса, домашній телефон, дата народження, дата вступу на роботу, зарплатня.

```
struct date
{ int day, month, year;           // День, місяць та рік
};

struct person
{ char prizv[20], adresa[150];    // Прізвище, адреса,
  char phone[10]; date birth_date; // телефон, дата народження,
```

```
    date work_date;           // дата вступу на роботу
    float salary;             // та зарплатня.
};
```

У цьому прикладі оголошується новий тип – структура `date` для зберігання дати (день, місяць, рік). Окрім того, оголошується тип – структура `person`, яка використовує тип `date` для полів `birth_date` і `work_date`. Посилання на поля вкладеної структури формується з імені структурної змінної, імені структурного поля й імені поля вкладеної структури. Те, що поля структур самі можуть бути структурами, надає можливість цілісно описувати доволі складні об'єкти реального світу. Позаяк структури, на відміну від масивів, зберігають дані різних типів, їх відносять до неоднорідних типів даних.

Оголосимо змінну створеного типу `person`:

```
person P;
```

Присвоювання прізвища у змінну `P`:

```
strcpy(P.prizv, "Шкуропат");
```

Присвоювання дати народження 5 березня 1987 року у змінну `P`:

```
P.birth_date.day=5;
P.birth_date.month=3;
P.birth_date.year=1987;
```

Розмістимо змінну `man` типу `person` у динамічній пам'яті:

```
person *man = new person;
```

І присвоїмо ті ж самі значення:

```
strcpy(man->prizv, "Шкуропат");
(man->birth_date).day=5;
(man->birth_date).month=3;
(man->birth_date).year=1987;
```

Оголосимо масив з даними про 100 осіб:

```
person P[100];
```

Попередні присвоювання для четвертої людини:

```
strcpy(P[3].prizv, "Шкуропат");
P[3].birth_date.day=5;
P[3].birth_date.month=3;
P[3].birth_date.year=1987;
```

Введемо значення всіх полів для всіх співробітників до масиву:

```
for (i=0; i<100; i++)
{ cout<<"Введіть прізвище: "
  gets(P[i].prizv);
  cout<<"Введіть адресу: "
  gets(P[i].adresa);
  cout<<"Введіть телефон: "
  gets(P[i].phone);
  cout<<"Введіть дату народження: "<<endl;
  cout<<"День: ";   cin>>P[i].birth_date.day;
```

```

    cout<<"Місяць: "; cin>>P[i].birth_date.month;
    cout<<"Рік: ";    cin>>P[i].birth_date.year;
    cout<<"Введіть дату вступу до роботи: "<<endl;
    cout<<"День: ";    cin>>P[i].work_date.day;
    cout<<"Місяць: "; cin>>P[i].work_date.month;
    cout<<"Рік: "      cin>>P[i].work_date.year;
}

```

Структурні змінні одного типу можна переприсвоювати одну одній, при цьому виконується присвоювання усіх полів, наприклад:

```
person man=P[0];
```

Як вже було зазначено вище, поля структури можуть бути якого завгодно типу, у тому числі масивами. Наприклад, оголошення структури «погода за місяць» може мати вигляд

```

struct pogoda
{ int month;          // Номер місяця,
  int temp[31];       // масив температур за 31 день
};
pogoda M;             // Оголошення змінної M типу pogoda (погода за 1 місяць)
pogoda G[12];         // і масиву G типу pogoda (погода за рік).

```

Визначимо значення температури повітря за 14 липня:

```

M.month = 7;
M.temp[13] = 27;

```

Присвоїмо випадкові значення температури за весь лютий:

```

M.month = 2;
for (int i=0; i<28; i++) M.temp[i] = random(5)-10;

```

Виведення температури повітря восени (за 3 місяці):

```

for (int i=9; i<=11; i++)
{ G[i-1].month = i;
  for(int j=0; j<31; j++)
    if(j==30 && (i==9 || i==11))
      continue;    // Пропустити 31.09 та 31.11
    else
      cout << G[i-1].temp[j] << " ";
  cout << endl;
}

```

У C++ структура при описуванні, окрім звичних полів, може містити елементи-функції. Наприклад, можна дописати до структури `pogoda` функцію `month_name`, яка визначатиме рядок – назву місяця за номером.

```

struct pogoda
{ int month, temp[31];    // Номер місяця і масив температур за 31 день
  char *month_name()
  { switch (month)
    { case 1: return "Січень";
      case 2: return "Лютий";
      case 3: return "Березень";
      case 4: return "Квітень";
    }
  }
};

```

```

        case 5: return "Травень";
        case 6: return "Червень";
        case 7: return "Липень";
        case 8: return "Серпень";
        case 9: return "Вересень";
        case 10: return "Жовтень";
        case 11: return "Листопад";
        case 12: return "Грудень";
        default: return "Помилковий номер";
    } }
};

```

У програмі цю функцію можна викликати в такий спосіб:

```

pogoda M;
. . .
cin >> M.month ;
cout << M.month_name();

```

Після виконання цих операторів на екран буде виведено рядок, утворений за допомогою функції `month_name()`.

Наведемо ще один приклад елемента-функції. Зорганізуємо структуру `student` з функцією `Show()`, яка будуватиме рядок класу `String`, поєднуючи усі поля цієї структури. Надалі цей рядок можна буде використовувати для виведення записів на екран:

```

ref struct student
{ String ^prizv, ^grupa;
  int Bal_math, Bal_inform;
  String^ Show()
  { return prizv+" "+ grupa+" "+Bal_math.ToString()+" "+
                                           Bal_inform.ToString();
  }
};

```

У програмі цю функцію можна викликати в такий спосіб:

```

student z;
z.prizv = "Іванов"; z.gr = "ПІ-11"; z.math = 60; z.inf = 90;
richTextBox1->Text = z.Show();

```

Після виконання цих операторів до елемента керування `richTextBox1` буде виведено рядок, утворений за допомогою функції `Show()`. Отже, звертання до елементів-функцій відбувається так само, як і до звичних полів. Наявність функцій у структурі може істотно спростити програму.

Дії над структурами

1) Для змінних одного й того самого структурного типу визначена *операція присвоювання*, при виконанні якої відбувається поелементне копіювання.

```

worker a, b = {"Хан", "менеджер", 23, 1750};
a = b;

```

Команда `a = b` скопіює відразу всі чотири поля структури `b` в поля структури `a`.

```

struct  tovar
{ char  nazva[12],  virobnik[10];  int  col;  float  vart;  } ;
tovar  x,  y;
scanf("%s %s %i %g", x.nazva, x.virobnik, &x.col, &x.vart);
y = x;
printf("%s %s %i %5.2f\n", y.nazva, y.virobnik, y.col, y.vart);

```

2) Структуру можна передавати до функції та повертати в якості результату роботи функції.

```

Show(worker z)
{ cout<<z.name<<" " <<z.position<<" " <<z.age<<" " <<z.salary<<endl;
}
worker Put()
{ worker w;
  cout<< "Введіть рядок з даними про робітника \n
          Прізвище Посада Вік Зарплатня" << endl;
  return cin >> w.name >> w.position >> w.age >> w.salary;
}
void main ()
{ date a, b = {"Лі", "бухгалтер", 33, 2700};
  Show(b); // виклик функції з передаванням структури b до функції
  a = Put(); // отримання структури в якості результату виконання функції
  Show(a);
  . . . . .
}

```

Приклад 10.1. Створити програму для опрацювання результатів сесії студентів: прізвище студента, група і результати двох екзаменів. У програмі передбачити можливість введення і виведення даних про довільну кількість студентів та відбір даних про студентів, які успішно здали сесію й вийшли на стипендію (не мають незадовільних оцінок, а середній бал склав понад 75 балів), також визначити прізвище студента з найбільшим середнім балом.

Розв'язок. У наведеному нижче програмному коді для одночасного введення всіх даних (чотирьох полів структури: прізвище, група і результати двох екзаменів) про одного студента використано функцію `scanf`, а виведення таких даних організовано за допомогою функції `printf`.

Ще однією особливістю цього програмного коду є почергове використання різних кодувань (команди `setlocale(0, ".1251")` і `setlocale(0, ".ОСР")`) для коректного виведення літер кирилиці в консолі. Без використання цих команд кирилиця виводитиметься у вигляді абракадабрі. Специфіка полягає у тому, що Visual C++ по-різному інтерпретує коди введених символьних даних і набраного у програмі тексту. Так, для виведення на екран текстових повідомлень слід використовувати кодування Windows 1251 (команда `setlocale(LC_ALL, ".1251")` або `setlocale(0, ".1251")`), а для виведення раніш введених символьних даних слід повернутися до початкових налаштувань командою `setlocale(0, ".ОСР")`.

Текст програмного коду:

```
#include <iostream>
using namespace std;
int main()
{
    setlocale(LC_ALL, ".1251");
    struct student
    { char surname[22], gr[8];
      int ex1, ex2;
    } ;
    int kol=0;           // загальна кількість студентів
    cout<<"Введіть кількість студентів - "; cin>> kol;
    student *z = new student[kol];
    cout<< "Введіть по чергово рядки з відомостями про успішність " << kol
         << " студентів: \nПрізвище Група Оцінка1 Оцінка2" << endl;
    for (int i=0; i<kol; i++)
        scanf("%s %s %i %i", z[i].surname, z[i].gr, &z[i].ex1, &z[i].ex2);
    cout<< "\n    Успішність " << kol
         << " студентів: \nПрізвище          Група Оцінка1 Оцінка2" << endl;
    setlocale(0, ".ОСР");
    for (int i=0; i<kol; i++)
        printf("%s\t%s\t%i\t%i\n", z[i].surname, z[i].gr, z[i].ex1, z[i].ex2);
    setlocale(0, ".1251");
    cout<< endl << "Успішно здали сесію та вийшли на стипендію:" << endl;
    int n=0;           // кількість стипендіатів
    double sr, max(0); // середній бал кожного та його максимальне значення
    char maxname[22];  // прізвище студента з максимальним середнім балом
    for (int i=0; i<kol; i++)
    { if (z[i].ex1>=60 && z[i].ex2>=60 && (z[i].ex1+z[i].ex2)/2.0 >= 75)
        { n++;
          setlocale(0, ".ОСР");
          printf("%s\t%s\t%i\t%i\n", z[i].surname, z[i].gr, z[i].ex1, z[i].ex2);
        }
        sr=(z[i].ex1+z[i].ex2)/2.0;
        if (sr>max) { max=sr; strcpy(maxname, z[i].surname); }
    }
    setlocale(0, ".1251");
    cout<< endl << "Кількість стипендіатів - " << n << endl;
    cout<< endl << "Максимальний середній бал " << max << " має ";
    setlocale(0, ".ОСР"); cout<< maxname << endl << endl;
    delete []z;
    system ("pause>>void");
    return 0;
}
```


Результати виконання програми:

```

Введіть кількість студентів - 3
Введіть по чергові рядки з відомостями про успішність 3 студентів:
Прізвище Група Оцінка1 Оцінка2
Костенко КТ-16 60 85
Халанчук ПІ-01 95 90
Мацкевич КТ-16 45 78

Успішність 3 студентів:
Прізвище Група Оцінка1 Оцінка2
Костенко КТ-16 60 85
Халанчук ПІ-01 95 90
Мацкевич КТ-16 45 78

Успішно здали сесію та вийшли на стипендію:
Халанчук ПІ-01 95 90

Кількість стипендіатів - 1
Максимальний середній бал 92.5 має Халанчук

```

Приклад 10.2. Створити програму для опрацювання даних телефонної книги (не менше п'яти абонентів): прізвище, ім'я, номер телефону, дата народження. Передбачити можливість виведення всіх даних телефонної книги та відбір даних про абонентів, номери яких розпочинаються з 067 або 068 (абоненти Київстар).

Розв'язок. У наведеному нижче програмному коді дані телефонної книги заповнюються самою програмою під час ініціалізації масиву структур.

Текст програмного коду:

```

#include <iostream>
#include <locale>
using namespace std;
int main()
{ setlocale(0, ".1251");
  struct abonent
  { char surname[21], name[15], tel[15], d[11];
  } z[]={ "Харитонов", "Іванна", "066-99-55-444", "11.05.1985",
          "Лавриненко", "Петро", "093-10-12-500", "23.12.87",
          "Кононова", "Олена", "067-55-10-123", "30.12.1990",
          "Петранчук", "Олег", "067-55-10-123", "03.02.56",
          "Холоденко", "Тетяна", "068-333-11-850", "15.08.79" };
  int kol = sizeof(z) / sizeof(abonent);
  cout<<" Телефонна книга з "<<kol
    <<" абонентами: \nПрізвище Ім'я Номер телефона Дата народження"<<endl;
  for(int i=0; i<kol; i++)
    cout<<z[i].surname<<"\t"<<z[i].name<<"\t"<<z[i].tel<<"\t"<<z[i].d<<endl;
  cout<<endl<<" Абоненти, номери яких розпочинаються з 067 або 068:"<<endl;
  for (int i=0; i<kol; i++)
    if (!strcmp(z[i].tel, "067", 3) || !strcmp(z[i].tel, "068", 3))
    { n++;
      cout<<z[i].surname<<"\t"<<z[i].name<<"\t"<<z[i].tel<<"\t"<<z[i].d<<endl;
    }
}

```

```
cout<<endl<<"Кількість абонентів Київстар - "<<n<<endl;
system ("pause">>void");
return 0;
}
```

Результати роботи:

```

Телефонна книга з 5 абонентами:
Прізвище      Ім'я      Номер телефону  Дата народження
Харитоновна   Іванна    066-99-55-444   11.05.1985
Лавриненко    Петро     093-10-12-500   23.12.87
Кононова      Олена     067-55-10-123   30.12.1990
Петранчук    Олег      067-55-10-123   03.02.56
Холоденко     Тетяна    068-333-11-850   15.08.79

Абоненти, номери яких розпочинаються з 067 чи 068:
Кононова      Олена     067-55-10-123   30.12.1990
Петранчук    Олег      067-55-10-123   03.02.56
Холоденко     Тетяна    068-333-11-850   15.08.79

Кількість абонентів Київстар - 3
```

Приклад 10.3. Ввести дані про товари: назва, фірма-виробник, кількість і ціна. Визначити сумарну вартість кожного з товарів фірми LG та середню вартість таких товарів.

Текст програмного коду:

```
#include <locale>
#include <iostream>
using namespace std;

int main()
{
    struct tovar
    { char nazva[12], pr[10]; int cnt; float vart;
    };
    int kol = 0;
    setlocale( LC_ALL, ".1251" );
    cout << "Введіть кількість товарів - "; cin >> kol;
    tovar *z = new tovar[kol];
    cout << "Введіть по чергово рядки з відомостями про " << kol
    << " товари(ів):\nНайменування Виробник Кіл-ть Ціна (десятькова кома)"<<endl;
    setlocale( LC_ALL, ".OCP" );
    for (int i=0; i<kol; i++)
        scanf("%s %s %i %g",z[i].nazva,z[i].pr,&z[i].cnt,&z[i].vart);
    double allsum=0, Svart, skol=0; int ktel=0;
    setlocale(LC_ALL, ".1251");
    cout << "\nПерегляд товарів фірми LG:
            \n№№ Найменування Виробник Кіл-ть Ціна Вартість" << endl;
    setlocale(LC_ALL, ".OCP");
    for (int i=0; i<kol; i++)
```

```

if (!strcmp(z[i].pr, "LG"))
{
    ktel++;
    skol += z[i].cnt;
    Svart = z[i].vart * z[i].cnt;
    allsum += Svart;
    printf("%i  %s\t%s\t%i\t%5.2f\t%6.2f\n",
           ktel, z[i].nazva, z[i].pr, z[i].cnt, z[i].vart, Svart);
}
allsum /= skol;
setlocale(LC_ALL, ".1251");
printf("\nСередня вартість товарів фірми LG - %5.2f\n", allsum);
delete []z;
system ("pause>>void");
return 0;
}

```

Результати роботи:

```

Введіть кількість товарів - 3
Введіть по черговому рядку з відомостями про 3 товари(iv):
Найменування Виробник Кіл-ть Ціна <десятькова кома>
Телефон LG 5 599,50
Телевізор Sony 3 4199,99
Планшет LG 10 2400,00

Перегляд товарів фірми LG:
## Найменування Виробник Кіл-ть Ціна Вартість
1 Телефон LG 5 599,50 2997,50
2 Планшет LG 10 2400,00 24000,00

Середня вартість товарів фірми LG - 1799,83

```

Приклад 10.4. Ввести відомості про прізвища і дати народження людей та визначити наймолодшого з них.

Розв'язок. Налаштування форми рекомендуємо провести у такій послідовності.

1) Розмістити на формі `dataGridView` і сформувати на ньому 2 стовпці (команда *Добавить столбец* у контекстному меню).

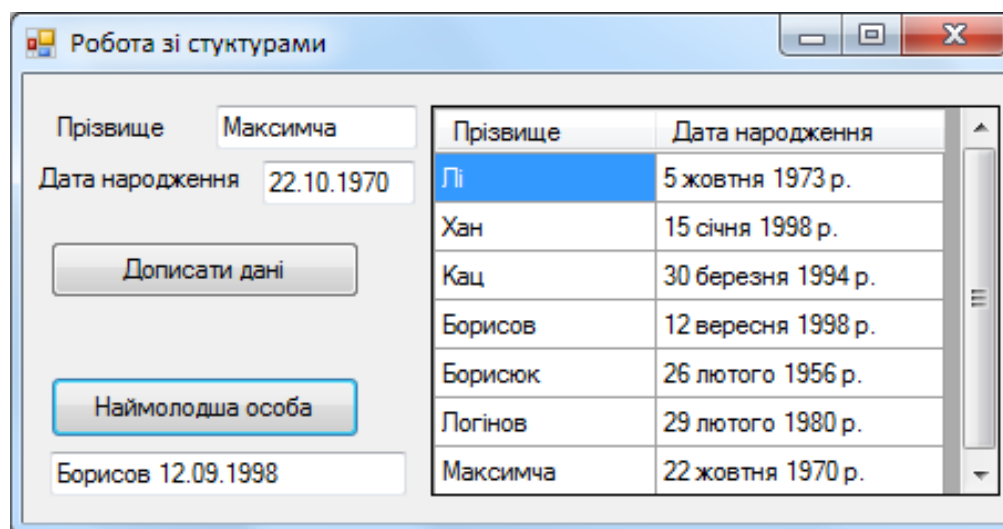
2) Командою контекстного меню *Правка столбцов* в однойменному діалоговому вікні для кожного стовпця задати текст заголовка (властивість `HeaderText`): *Прізвище* і *Дата народження*.

3) Задати для властивостей `AllowUserToAddRows` (дозвіл автоматичного додавання рядків) і `RowHeadersVisible` (відображення заголовка рядків) значення `False`.

4) Розмістити на формі два елементи `label`, два `textbox` і дві кнопки `button`. Задати надписи на них (властивість `Text`):

- для `Form1` – *Робота зі структурами*;
- для `label1` – *Прізвище*;
- для `label2` – *Дата народження*;

- для button1 – *Дописати дані*;
- для button2 – *Наймолодша особа*.



Текст програми:

```
ref struct person // посилальний тип структури з двома полями
{ String^ prizm;
  DateTime birth;
};
// глобальна змінна – кількість осіб, на початку має значення 0 і буде
// збільшуватись на 1 при натисканні кнопки «Дописати дані»
static int n=0;
// Кнопка «Дописати дані»
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    person p;
    p.prizm = textBox1->Text;
    p.birth = DateTime::Parse(textBox2->Text);
    dataGridView1->Rows->Add();
    dataGridView1[0,n]->Value = p.prizm;
    dataGridView1[1,n]->Value = p.birth.ToString();
    n++;
}
// Кнопка «Наймолодша особа»
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e)
{
    int k=dataGridView1->Rows->GetRowCount(DataGridViewElementStates::Visible);
    person p;
    String^ young; DateTime d, max=DateTime::Parse("01.01.1900");
    // або max=DateTime::Parse(dataGridView1[1,0]->Value->ToString());
    for (int i=0; i<k; i++)
    { d = DateTime::Parse(dataGridView1[1,i]->Value->ToString());
      // або young = Convert::ToString(dataGridView1[0,i]->Value);
      if (max < d)
      { young = dataGridView1[0,i]->Value->ToString();

```

```
    max = d;
  }
}
textBox3->Text = young + " " + max.ToShortDateString();
}
```

Структури дозволяють описати майже всі навколишні об'єкти. Однак існують певні винятки. Наприклад, припустімо, що структура `person` має для чоловіків містити найменування військової спеціальності, а для жінок – кількість дітей. Розв'язок «у лоба» може мати вигляд

```
struct person
{ date birth;           // Дата народження,
  char sex;             // стать (ч/ж),
  char wars[20];        // військова спеціальність
  int kids;             // і кількість дітей.
};
```

Синтаксично начебто все правильно, але виникає можливість описати чоловіка, який народжував дітей (на жінок-військових ми покищо не зважаємо). Щоб уникнути такого становища, слід оголосити поля `wars` і `kids` у такий спосіб, щоб вони не могли мати значення водночас, тобто об'єднати ці поля в одне ціле. Для цього в C++ існують поєднання (`union`), які докладніше буде розглянути у наступному підрозділі:

```
struct person
{ date birth;
  char sex;
  union { char wars[20]; int kids; };
};
```

При використанні об'єднань водночас розв'язується й завдання економії пам'яті. Однак комп'ютер не відстежує, яке саме з двох полів, `wars` чи `kids`, має існувати даного моменту. Для компілятора водночас існують обидва поля.

10.4 Перерахування (`enum`)

При написанні програм часто виникає потреба у визначенні наперед відомої кількості іменованих констант, які мають мати різні значення (при цьому конкретні значення можуть бути неважливими). Для цього зручно користуватися типом перерахування `enum`, всі можливі значення якого задаються списком цілочисельних констант:

```
enum [ <ім'я_типу> ] { <список_констант> };
```

Ім'я типу задається в тому разі, якщо у програмі потрібно визначати змінні цього типу. Змінним перераховного типу можна присвоювати кожне значення зі списку констант, зазначених при оголошуванні типу. Імена перераховних констант мають бути унікальними. Перераховні константи подаються й опрацьовуються як цілі числа і можуть ініціалізуватися у звичайний спосіб. Окрім того вони можуть мати однакові значення. За відсутності ініціалізації перша

константа обнулюється, а кожній наступній присвоюється значення на одиницю більше, ніж попередній.

```
enum week { sat=0, sun=0, mon, tue, wed, thu, fri } rob_den;
```

Тут описано перерахування `week` з відповідною множиною значень і оголошено змінну `rob_den` типу `week`. Перераховні константи `sat` та `sun` мають значення 0, `mon` – значення 1, `tue` – 2, `wed` – 3, `thu` – 4, `fri` – 5.

До перераховних змінних можна застосовувати арифметичні операції й операції відношення, наприклад:

```
enum days {sun, mon, tue, wed, thu, fri, sat};
days day1 = mon, day2 = thu;
int diff = day2 - day1;
cout << "Різниця у днях: " << diff << endl;
if (day1 < day2)
    cout << "day1 настане раніш, ніж day2 \n";
```

Арифметичні операції над змінними перераховних типів зводяться до операцій над цілими числами. Проте, не зважаючи на те, що компіляторі відомо про цілочисельну форму подання перераховних значень, варто використовувати цей факт надто обережно. Якщо спробувати виконати присвоєння

```
day1 = 5;
```

компілятор видасть повідомлення (хоча компіляція відбудеться без помилок). Рекомендовано без нагальної потреби не використовувати цілочисельну інтерпретацію перераховних значень.

Значення перераховним константам можна встановлювати явно, ініціалізуючи їх при оголошенні, причому значення мають бути лише цілими числами, наприклад:

```
enum {first, second = 100, third};
```

У цьому разі `first` за промовчаням дорівнює 0, а наступні неініціалізовані константи перерахування перевищують своїх попередників на 1. Наприклад, `third` матиме значення 101.

Істотним недоліком типів перерахування є те, що вони не розпізнаються засобами введення-виведення C++. При виведенні такої змінної виводиться не її формальне значення, а її внутрішнє подання, тобто ціле число. Наведемо приклад програми, яким можна скористатись за потреби виведення значень таких змінних у зрозумілому для сприйняття вигляді.

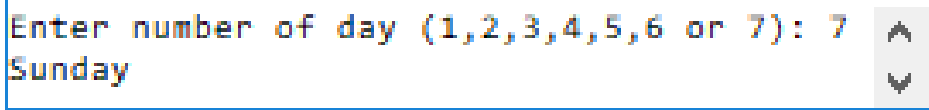
```
#include <iostream>
#include <string>
using namespace std;

enum day {mon=1, tue, wed, thu, fri, sat, sun};
string dayToString(const day& one)
{
    string arr[]={ "Monday", "Tuesday", "Wednesday",
                  "Thursday", "Friday", "Saturday", "Sunday" };
    return arr[one-1];
}
```

```

day intToDay(int one)
{ return static_cast<day>(one);
}
int main()
{ system("color F0");
  int numb;
  cout << "Enter number of day (1,2,3,4,5,6 or 7): ";
  cin >> numb;
  day one = intToDay(numb);
  cout << dayToString(one) << '\n';
  cin.get();
  return 0;
}

```



Використаний у функції `intToDay()` оператор `static_cast` має такий формат:

`static_cast < тип > ( вираз )`

і перетворює без жодних перевірок відповідності під час виконання програми *вираз* до зазначеного *типу*. У нашому разі відбуватиметься перетворення введеної цілої змінної `one` до перерахованого типу `day`.

Питання та завдання для самоконтролю

- 1) В який спосіб можна створювати нові типи даних на базі вже наявних?
- 2) Коли використовується службове слово `typedef`?
- 3) Записати оголошення типу `Str` як масиву з 60-ти символів.
- 4) Оголосити тип як матрицю дійсних чисел розмірністю 7×7.
- 5) Дати означення типу даних «об'єднання».
- 6) Як визначається обсяг пам'яті, потрібний для зберігання усіх полів певного об'єднання?
- 7) Написати об'єднання, в якому може зберігатися чи то вказівник, чи ціле, чи дійсне число.
- 8) Дати означення структури як типу даних.
- 9) У чому схожість і відмінність опису і специфіки використання об'єднання і структури?
- 10) У чому полягає відмінність структури від масиву?
- 11) Як здійснюється доступ до полів структури?
- 12) Як визначається обсяг пам'яті, потрібний для зберігання структури?
- 13) Чи можуть в одній програмі збігатися імена полів структури і змінних?
- 14) Чи можуть збігатися в одній програмі імена полів двох структур?
- 15) Чи дозволяється створювати вкладені структури?

16) Оголосити структуру, яка описуватиме для деякого електричного приладу такі характеристики: назва приладу, споживана потужність і номінальна напруга.

17) Чи може матриця бути полем структури? Якщо так, навести приклад.

18) Чи має значення порядок слідування полів у описі структури?

19) Чи може структура бути типом елемента масиву? Якщо так, навести приклад.

20) Оголосити структуру з ім'ям `Student`, що містить такі поля: прізвище, назва групи, масив з п'яти екзаменаційних оцінок. Записати команди, які нададуть початкові значення усім полям змінної типу `Student`.

21) Знайти та виправити помилки припущені в описі структури:

```
structure { int arr[12], char string, int *sum }
```

22) Чи може полем структури бути функція?

23) Чи можна структуру передавати до функції аргумента функції?

24) Які дії можна виконувати над структурами?

25) Охарактеризувати тип перерахування.

26) Яких значень набувають константи перерахування за відсутності їхньої ініціалізації?

27) Якщо перерахування `COLOR` задано в такий спосіб, то яким є значення його елементів `white`, `red`, `blue` та `yellow`?

```
enum COLOR { white, black=100, red, blue, green=300, yellow};
```

28) Які помилки містять наведені оператори?

```
enum State { on, off };
```

```
enum YesNo { on, off };
```

29) Правильним чи ні є таке оголошення перерахування?

```
enum YesNo { no = 0, No = 0, yes = 1, Yes = 1 };
```

30) Створити оголошення перерахування `response` зі значеннями «Так», «Ні» і «Можливо». Значення «Так» повинно мати порядковий номер 1, «Ні» – 0 та «Можливо» – 2.

РОЗДІЛ 11

ТИПИ ОПРАЦЮВАННЯ ДАТИ І ЧАСУ

11.1 Тип «дата-час» у C++

Календарний час містить в собі рік, місяць, день місяця, години, хвилини, секунди. Функції й типи даних, необхідні для опрацювання дати і часу, оголошено у заголовному файлі `time.h`. Зокрема цей файл містить визначення типів даних `time_t`:

```
typedef long time_t;
typedef long clock_t;
```

і структури `tm`, оголошеної в такий спосіб:

```
struct tm
{
    int    tm_sec;    // секунди 0..60
    int    tm_min;    // хвилини 0..59
    int    tm_hour;   // години 0..23
    int    tm_mday;   // день місяця 1..31
    int    tm_mon;    // місяць 0..11 (0=січень)
    int    tm_year;   // кількість років від 1900 р.
    int    tm_wday;   // день тижня 0..6 (0=неділя)
    int    tm_yday;   // день року 0..365
    int    tm_isdst;  // літній час (булеве значення)
};
```

Час у форматі `time_t` по суті є числом – кількістю секунд, рахуючи від 00:00 годин 1 січня 1970 р. (unix timestamp).

Час у форматі `struct tm` подано структурою з полями: секунди, хвилини, години, день, місяць, рік тощо. Звертаємо увагу, що рік рахується від 1900, тобто 2018 р. має зберігатися як 118, а місяць рахується від 0, отже квітень це 3.

Основні функції для опрацювання дати і часу з описом їхньої роботи наведено у табл. 11.1.

Таблиця 11.1

Функції опрацювання дати і часу

Синтаксис функції	Опис роботи функції
<code>char *asctime (struct tm time);</code>	Перетворює дату і час <code>time</code> з формату структури типу <code>tm</code> на символічний рядок з 26-ти символів, який може мати вигляд Mon Jan 04 02:03:55 2010\n\0 Функція повертає вказівник на цей рядок
<code>clock_t clock();</code>	Повертає процесорний час, який сплинув від початку виконання програми, чи <code>-1</code> , якщо він є невідомий

Синтаксис функції	Опис
<code>char *ctime(long *time);</code>	Перетворює час <code>time</code> формату <code>long*</code> у символічний рядок з 26 символів, який може виглядати: Mon Jan 04 02:03:55 2010\n\0 Функція повертає вказівник на цей рядок
<code>double difftime (time_t time2, time_t time1);</code>	Повертає різницю поміж <code>time2</code> і <code>time1</code> типу <code>time_t</code> , тобто час у секундах, який сплине від <code>time2</code> до <code>time1</code> , як число з подвійною точністю
<code>struct tm *gmtime (long *time);</code>	Перетворює <code>time</code> типу <code>long*</code> до формату структури <code>tm</code>
<code>struct tm *localtime (long *time);</code>	Перетворює <code>time</code> типу <code>long*</code> до формату структури <code>tm</code> з урахуванням зони місцевого часу, якщо попередньо було задано глобальну змінну часової зони <code>TZ</code>
<code>time_t mktime (struct tm *time);</code>	Перетворює час і дату <code>time</code> формату структури <code>tm</code> до формату <code>time_t</code>
<code>char * _strdate (char *date);</code>	Перетворює поточну дату на символічний рядок <code>time</code> з 9 символів формату <code>MM/DD/YY</code>
<code>char * _strtime (char *time);</code>	Перетворює поточний час на символічний рядок <code>time</code> з 9 символів формату <code>HH:MM:SS</code>
<code>size_t strftime(char *s, size_t maxsize, char *fmt, struct tm *t);</code>	Перетворює дату і час <code>t</code> формату структури <code>tm</code> на символічний рядок <code>s</code> розміром <code>maxsize</code> у форматі <code>fmt</code>
<code>time_t time (time_t *timer);</code>	Повертає поточний календарний час (тобто час в секундах, який сплинув з півночі (00:00:00) 1 Січня 1970 за Грінвічем)
<code>void tzset();</code>	Функція встановлює значення глобальних змінних <code>daylight</code> (ненульове значення типу <code>int</code> означає дозвіл переходу на літній час), <code>timezone</code> (значення типу <code>long</code> задає різницю в секундах поміж місцевим часом і часом за Грінвічем (GMT)) і <code>tzname</code> (рядок типу <code>*char</code> трілітерних часових зон)

Розглянемо докладніше специфікатори формату для функції `strftime`, яка перетворює дату і час `t` формату структури `tm` на символічний рядок `s` розміром `maxsize` у форматі `fmt` і зазвичай використовується для виведення дати, поданої у форматі `struct tm`. Ця функція має чотири параметри: `s`, `maxsize`, `fmt`, `t`. До рядка `s` записується результат виконання функції. Число `maxsize` задає максимальну довжину рядка-результату. Формат `fmt` – це рядок, який містить будь-яку комбінацію специфікаторів, які починаються зі знаку `%`. Функція перетворює ці специфікатори на відповідні рядкові значення. Деякі зі специфікаторів формату, які використовуються у функції `strftime`, подано у табл. 11.2.

Специфікатори формату для функції strftime

Специфікатор	Значення специфікатора	Приклад
%a	Скорочена назва дня тижня*	Thu
%A	Повна назва дня тижня*	Thursday
%b	Скорочена назва місяця*	Aug
%B	Повна назва місяця*	August
%c	Дата і час*	08/23/01 14:55:02
%d	День місяця у вигляді двоцифрового числа (01-31)	23
%D	Скорочена дата у форматі MM/DD/YY, еквівалентна до %m/%d/%y	08/23/01
%e	День місяця, в одноцифрових числах перший знак замінено на пробіл (1-31)	23
%H	Години у 24-годинному форматі (00-23)	14
%I	Години у 12-годинному форматі (01-12)	02
%m	Місяць у числовому вигляді (01-12)	08
%M	Хвилини (00-59)	55
%p	АМ або РМ	PM
%r	Час у 12-годинному форматі*	02:55:02 pm
%R	Час у 24-годинному форматі HH:MM, еквівалентно до %H:%M	14:55
%S	Секунди (00-61)	02
%w	Номер дня тижня, неділя = 0 (0-6)	4
%W	Номер тижня з першим понеділком як першим днем тижня (00-53)	34
%x	Дата*	08/23/01
%X	Час*	14:55:02
%y	Останні дві цифри року (00-99)	01
%Y	Рік	2001

* Специфікатори, позначені зірочкою (*), залежать від локалізації.

Приклад 11.1. Вивести поточну системну дату і час.

Текст програми:

```
#include <iostream>
#include <time.h>
using namespace std;
int main ()
{ time_t t;           // Оголошення змінної t
  t = time(NULL);     // Запис поточного часу до t
  printf("%s\n", ctime(&t));
  system("pause");
  return 0;
}
```

Результат виконання програми:

Mon Jun 25 16:33:16 2018

Приклад 11.2. Проілюструємо роботу деяких з функцій, наведених у табл. 11.1, на прикладі консольної програми для виведення поточної дати і часу у заданому форматі. Час виводитиметься у 12-годинному форматі.

Текст програми:

```
#include <iostream>
#include <time.h>
using namespace std;
int main()
{
    struct tm *newtime;    char *am_pm="PM";
    time_t long_time; // Оголошення змінної long_time.
    time(&long_time); // Запис поточного часу до long_time.
    // Перетворення long_time на структуру newtime.
    newtime=localtime(&long_time);
    if (newtime->tm_hour<12) // Якщо проминуло менше за 12 годин доби,
        am_pm="AM";    // відбудеться відповідне визначення рядку формату.
    if (newtime->tm_hour>12) // Якщо проминуло більше за 12 годин доби,
        newtime->tm_hour-=12; // відбудеться віднімання 12.
    printf("%.20s %s\n", asctime(newtime), am_pm);
    cin.get();
    return 0;
}
```

Результати виконання програми:

Fri Jun 15 08:17:55 PM

Приклад 11.2. Проілюструємо роботу функції `strftime`, яка перетворює введені значення дати і часу (формату структури `tm`) на символічний рядок у заданому форматі день/місяць/рік.

Текст програми:

```
#include <iostream>
#include <time.h>
using namespace std;
int main ()
{
    setlocale(0, ".1251");
    time_t t;
    struct tm tmstr;
    int day, month, year;
    char str[256];
    cout<<"Введіть день: "; cin>>day; tmstr.tm_mday = day;
    cout<<"Введіть місяць: "; cin>>month;
    tmstr.tm_mon = month-1; // Позаяк нумерація місяців починається з 0
```

```

cout<<"Введіть рік: "; cin>>year;
tmstr.tm_year = year-1900; // Оскільки роки починаються з 1900
tmstr.tm_hour = 0;   tmstr.tm_min = 0;   tmstr.tm_sec = 0;
// Виведення tmstr у форматі "день/місяць/рік"):
strftime ( str, sizeof(str), "%d/%m/%Y", &tmstr );
cout<<str<<endl;
system("pause");
return 0;
}

```

Результати виконання програми:

```

Введіть день: 25
Введіть місяць: 6
Введіть рік: 2018
25/06/2018

```

Наведений вище приклад 11.2 можна реалізувати дещо інакше:

```

#include <iostream>
#include <time.h>
using namespace std;
int main()
{ struct tm *time_now;  time_t s_now;
  char str[80];
  tzset();
  time(&s_now);
  time_now = localtime(&s_now);
  strftime(str,80,"It is %M minutes after %I o'clock (%Z)  %A, %B %d 20%y",
           time_now);
  printf("%s\n",str);
  cin.get();
  return 0;
}

```

Результати виконання програми:

```

It is 32 minutes after 08 o'clock (Фінляндія (літо)) п'ятниця, червень 15 2018

```

Вводити дату і час можна так, як показано у прикладі 11.3 (окремо день, місяць, рік, години, хвилини, секунди). Але інколи зручніше задавати дату (час) у вигляді рядку певного формату, який треба спочатку розбити на день, місяць та рік (години, хвилини, секунди) за допомогою, наприклад, функції `sscanf`. Розглянемо це на прикладі.

Приклад 11.3. Дату і час, записані у вигляді рядків у форматі "тільки дата", "тільки час" та "дата і час", перетворити на тип `time_t`.

Текст програми:

```

#include <iostream>
#include <time.h>
using namespace std;

```

```

int main ()
{
    setlocale(0, ".1251");
    char *sd="12/04/2005";
    char *st="12:04:05";
    char *sdt="12/04/2005 12:04:05";
    struct tm tmstr;
    puts(sd);
    sscanf(sd, "%d/%d/%d", &tmstr.tm_mday, &tmstr.tm_mon, &tmstr.tm_year);
    tmstr.tm_mon-=1; tmstr.tm_year-=1900;
    // Позаяк час не задано, задаємо нулі замість годин, хвилин і секунд
    tmstr.tm_hour=0; tmstr.tm_min=0; tmstr.tm_sec=0;
    time_t t1=mktime (&tmstr); printf("%s\n", ctime(&t1));
    puts(st);
    sscanf(st, "%d:%d:%d", &tmstr.tm_hour, &tmstr.tm_min, &tmstr.tm_sec);
    // Замість дати задаємо будь-яку, наприклад, 01.01.2000
    tmstr.tm_mday=1; tmstr.tm_mon=0; tmstr.tm_year=100;
    t1=mktime (&tmstr); printf("%s\n", ctime(&t1));
    puts(sdt);
    sscanf(sdt, "%d/%d/%d %d:%d:%d", &tmstr.tm_mday, &tmstr.tm_mon,
        &tmstr.tm_year, &tmstr.tm_hour, &tmstr.tm_min, &tmstr.tm_sec);
    tmstr.tm_mon-=1; tmstr.tm_year-=1900;
    t1=mktime (&tmstr);
    printf("%s\n", ctime(&t1));
    system("pause");
    return 0;
}

```

Результат виконання програми:

```

12/04/2005
Tue Apr 12 00:00:00 2005

12:04:05
Sat Jan 01 11:04:05 2000

12/04/2005 12:04:05
Tue Apr 12 13:04:05 2005

```

Приклад 11.4. Ввести дату народження людини і визначити її вік (кількість повних років).

Розв'язок. Звертаємо увагу, що для дат до 01.01.1970 р. тип `time_t` використовувати не можна. Тому для обчислення віку в секундах потрібно розрахувати окремо проміжок у секундах між поточною датою та днем народження у 1970 році й окремо кількість секунд у роках між роком народження і 1970.

Текст програми:

```

#include <iostream>
#include <time.h>
using namespace std;

```



```
int main ()
{ setlocale(0, ".1251");
  char birthday[30];
  time_t t, t1;
  struct tm tmstr;
  int year, seconds, yearinseconds;
  yearinseconds=(int)365.25*24*60*60;
  t = time(NULL); // поточна дата
  cout<<"Введіть дату народження: ";
  gets_s(birthday,30);
  sscanf_s(birthday, "%d/%d/%d", &tmstr.tm_mday, &tmstr.tm_mon,
            &tmstr.tm_year);

  tmstr.tm_mon-=1; tmstr.tm_year-=1900;
  tmstr.tm_hour=0; tmstr.tm_min=0; tmstr.tm_sec=0;
  if (tmstr.tm_year>=70)
  { t1=mktime(&tmstr); seconds=difftime(t,t1); }
  else
  {
    year=70-tmstr.tm_year; tmstr.tm_year=70;
    t1=mktime(&tmstr);
    seconds=difftime(t,t1)+(int)year*365.25*24*60*60;
  }
  cout<<"Кількість повних років = "<<seconds/yearinseconds<<endl;
  system("pause");
  return 0;
}
```

Результати виконання програми:

1) Дата народження після 1970 року:

Введіть дату народження: 12/07/1998
Кількість повних років = 19

2) Дата народження до 1970 року:

Введіть дату народження: 12/07/1965
Кількість повних років = 52

Приклад 11.5. Ввести час початку кіносеансу і визначити час його завершення, якщо тривалість показу 1 година 45 хвилин.

Розв'язок. Введемо час початку сеансу у форматі чч:мм у рядкову змінну `st_time`, а час закінчення сеансу – у рядкову змінну `end_time`.

Структура `tmstr` містить поточну дату, години і хвилини (початок кіносеансу), які вводяться з клавіатури, а `endstr` вказує на час закінчення сеансу.

Текст програми:

```
#include <iostream>
#include <time.h>
using namespace std;
```

```

int main ()
{ setlocale(0, ".1251");
  char st_time[30], end_time[30];
  time_t end_t, t, t1;
  struct tm tmstr, *endstr;
  t = time(NULL);
  tmstr=*localtime(&t);
  cout<<"Введіть час початку сеансу: ";
  gets_s(st_time,30);
  sscanf_s(st_time, "%d:%d", &tmstr.tm_hour, &tmstr.tm_min);
  t1=mktime (&tmstr);
  end_t=t1+(45+60)*60;
  endstr=localtime(&end_t);
  strftime(end_time,30,"%H:%M", endstr);
  puts(end_time);
  system("pause");
  return 0;
}

```

Результат виконання програми:

Введіть час початку сеансу: 15:40

Час закінчення сеансу: 17:25

Приклад 11.6. Ввести дату виготовлення продукту і термін його зберігання. Визначити, чи закінчився його термін придатності.

Текст програми:

```

#include <iostream>
#include <time.h>
using namespace std;
int main ()
{ setlocale(0, ".1251");
  char st_time[30], term[30];
  time_t t, t1, end_t;
  struct tm tmstr, termstr;
  int d, m, y, iterm;
  t = time(NULL);
  cout<<"Введіть дату виготовлення: ";
  gets_s(st_time,30);
  cout<<"Введіть термін придатності: ";
  gets_s(term,30);
  sscanf_s(st_time, "%d/%d/%d", &tmstr.tm_mday, &tmstr.tm_mon,
                                                    &tmstr.tm_year);
  tmstr.tm_mon-=1; tmstr.tm_year-=1900;
  tmstr.tm_hour=0; tmstr.tm_min=0; tmstr.tm_sec=0;
  sscanf_s(term, "%d/%d/%d", &d, &m, &y);
  iterm=(int)(y*365.25+m*30+d)*24*60*60;
  t1=mktime (&tmstr);

```

```
end_t=t1+iterm;  
printf("Кінцевий термін: %s\n", ctime(&end_t));  
if(t>end_t)  
    cout<<"Термін зберігання закінчився"<<endl;  
else  
    cout<<"Термін зберігання ще не закінчився"<<endl;  
system("pause");  
return 0;  
}
```

Результати виконання програми:

1) Термін зберігання 5 днів:

Введіть дату виготовлення: 12/03/2018
Введіть термін придатності: 5/0/0
Кінцевий термін: Sat Mar 17 00:00:00 2018

Термін зберігання закінчився

2) Термін зберігання 3 місяці:

Введіть дату виготовлення: 12/03/2018
Введіть термін придатності: 0/3/0
Кінцевий термін: Sun Jun 10 01:00:00 2018

Термін зберігання закінчився

3) Термін зберігання 2 роки:

Введіть дату виготовлення: 12/03/2018
Введіть термін придатності: 0/0/2
Кінцевий термін: Wed Mar 11 00:00:00 2020

Термін зберігання ще не закінчився

11.2 Дата-час у Visual C++

11.2.1 Типи даних

Платформа .NET Framework опрацювання дати і часу має спеціальні типи-структури: `DateTime`, `DateTimeOffset`, `TimeSpan` та ін.

DateTime⁹ – базовий засіб для роботи з датою і часом, який дозволяє визначити конкретні дату і час доби, при чому можна працювати або лише з датою – «27.11.2014», або лише з часом – «11:25:00», або і з датою і з часом водночас – «27.11.2014 11:25:00». З датами і часом можна виконувати арифметичні дії, наприклад, додати до конкретної дати шість місяців. Змінна такого типу займає 8 байтів. `DateTime` подає дату й час у діапазоні від 00:00:00 (північ) 1 січня 0001 року (н. е.) до 23:59:59 31 грудня 9999 року за григоріанським календарем.

⁹ Більш докладно зі структурою `DateTime` та її методами можна ознайомитись за посиланням: [https://msdn.microsoft.com/ru-ru/library/windows/desktop/system.datetimeoffset\(v=vs.90\).aspx](https://msdn.microsoft.com/ru-ru/library/windows/desktop/system.datetimeoffset(v=vs.90).aspx).

Значення часу вимірюються в 100-наносекундних одиницях, званих тактами, і точна дата подається числом тактів з 00:00 1 січня 0001 року нашої ери. Наприклад, значення тактів 312413760000000000L задає дату 1 січня 0100 року.

Тип `DateTime` специфічний тим, що значення дати і часу однозначно ідентифікує конкретний момент часу в конкретному часовому поясі і неоднозначно веде себе при зміні часових поясів (в такому разі використовують `DateTimeOffset`).

Тип `DateTimeOffset`¹⁰ дозволяє однозначно визначати момент часу з урахуванням часового поясу, тобто `DateTimeOffset` складається зі значення `DateTime`, доповненого властивістю `Offset` (час у форматі UTC). Змінна такого типу займає 12 байтів. Значення `DateTimeOffset` не пов'язане з конкретним часовим поясом, але може бути створене з будь-якого часового поясу. Тип `DateTimeOffset` доречний для використання, коли треба однозначно ідентифікувати конкретний момент часу, наприклад, при веденні журналу часу транзакцій чи подій системи, особливо коли припускається перенесення даних з одного ПК на інший. Тип `DateTimeOffset` включає більшість функцій типу `DateTime`, але не замінює цей тип. Використання значень `DateTimeOffset` більш поширене, аніж `DateTime`, і розглядається як тип «дата-час» за промовчанням для розроблення програм.

З певним значенням `DateTimeOffset` можна виконувати операції додавання і віднімання, додаючи і віднімаючи будь-які інтервали дати і часу. В арифметичних операціях зі значеннями `DateTimeOffset`, на відміну від арифметичних операцій над значеннями `DateTime`, при поверненні результату враховується поправка на часовий пояс.

Тип `TimeSpan`¹¹ призначений для роботи з інтервалом часу (тривалістю часу або витраченим часом), який вимірюється як додатне або від'ємне число днів, годин, хвилин і секунд ($\pm$ д.чч:мм:сс). Змінна такого типу займає 8 байтів. Максимальною одиницею часу, яку використовує `TimeSpan` для вимірювання тривалості, є день. Значення `TimeSpan` об'єкт являє кількість тактів, рівне представленому інтервалу часу. Такт дорівнює 100 наносекунд або однієї десятимілльйонна секунди. Інтервали часу змінюються днями для узгодженості, позаяк кількість днів у більш великих одиницях часу, наприклад, місяцях чи то роках, варіюється. `TimeSpan` може також використовуватися для подання часу дня, але тільки якщо час не пов'язаний з визначеною датою. Інакше слід використовувати `DateTime` або `DateTimeOffset`.

У табл. 11.3 наведено методи перетворювання дати і часу на рядок та навпаки.

¹⁰ Більш докладно зі структурою `DateTimeOffset` можна ознайомитись за посиланням: [https://msdn.microsoft.com/ru-ru/library/system.datetimeoffset\(v=vs.110\).aspx](https://msdn.microsoft.com/ru-ru/library/system.datetimeoffset(v=vs.110).aspx).

¹¹ Більш докладно зі структурою `TimeSpan` та її методами можна ознайомитись за посиланням: [https://msdn.microsoft.com/ru-ru/library/system.timespan\(v=vs.110\).aspx](https://msdn.microsoft.com/ru-ru/library/system.timespan(v=vs.110).aspx).

Методи перетворювання дати і часу

Синтаксис методів	Опис
ToLocalTime()	Перетворює значення об'єкта DateTime на рядок зі значенням місцевого часу
ToLongDateString()	Перетворює значення DateTime на рядок з «довгим» значенням дати
ToLongTimeString()	Перетворює значення DateTime на рядок з «довгим» значенням часу
ToShortDateString()	Перетворює значення DateTime на рядок з «коротким» значенням дати
ToShortTimeString()	Перетворює значення DateTime на рядок з «коротким» значенням часу.
Parse(String^); Parse(String^, IFormatProvider^); Parse(String^, IFormatProvider^, DateTimeStyles)	Перетворюють рядок на значення «дата-час» типу DateTime з урахуванням формату і регіональних параметрів
ToString(); ToString(String^); ToString(String^, IFormatProvider^)	Перетворюють числове значення поточного об'єкта TimeSpan або DateTime на рядок з використанням заданого формату
TryParse(String^, TimeSpan%); TryParse(String^, IFormatProvider^, TimeSpan%)	Перетворюють рядок на інтервал часу TimeSpan з урахуванням формату і регіональних параметрів
TryParse(String^, DateTime%); TryParse(String^, IFormatProvider^, DateTimeStyles, DateTime%)	Перетворюють рядок на подання дати і часу в DateTime

У методах перетворення типів з табл. 11.3, використовується рядок форматування дат. У табл. 11.4 наведено значення специфікаторів формату дат.

Таблиця 11.4

Специфікатори формату дат

Специфікатор	Опис
d	Виведення дня без початкових нулів (1 – 31)
dd	Виведення дня з початковим нулем (01 – 31)
ddd	Виведення дня у вигляді позначення (пн – нд)
dddd	Виведення дня у вигляді повної назви дня тижня (понеділок – неділя)
М	Виведення місяця без початкових нулів (1 – 12)
ММ	Виведення місяця з початковими нулями (01 – 12)
МММ	Виведення місяця у вигляді позначень (січ – грд)
ММММ	Виведення повної назви місяця (січень – грудень)
уу	Виведення двох останніх цифр року (00 – 99)
уууу	Виведення року у повному форматі (0000 – 9999)

Специфікатор	Опис
H	Виведення годин без початкового нуля (0 – 23)
HH	Виведення годин з початковим нулем (00 – 23)
h	Виведення годин без початкового нуля (0 – 11)
hh	Виведення годин з початковим нулем (00 – 11)
m	Виведення хвилин без початкового нуля (0 – 59)
mm	Виведення хвилин з початковим нулем (00 – 59)
s	Виведення секунд без початкового нуля (0 – 59)
ss	Виведення секунд з початковим нулем (00 – 59)

Покажемо різні можливості специфікаторів форматного виведення значення дати і часу.

```
DateTime d=DateTime::Now;
richTextBox2->Text += d.ToString()+"\n";           // 05.11.2018 8:45:00
richTextBox2->Text += d.ToString("g")+"\n";         // 05.11.2018 8:45
richTextBox2->Text += d.ToString("t")+"\n";         // 8:45 – лише час
richTextBox2->Text += d.ToString("d")+"\n";         // 05.11.2018 – лише дата
richTextBox2->Text += d.ToString("dddd")+"\n";       // понеділок
richTextBox2->Text += d.ToString("MM/dd/yy HH:mm")+"\n"; // 11.05.18 08:45
richTextBox2->Text += d.ToString("dd/MM/yy dddd")+"\n"; // 05.11.2018 понеділок
richTextBox2->Text += d.ToString("dddd d MMMM yууу HH:mm zzz")+"\n";
// понеділок 5 листопада 2018 8:45 +02:00
```

11.2.2 Основні засоби для опрацювання дати і часу

Деякі конструктори

DateTime(Int32, Int32, Int32) – ініціалізує новий екземпляр структури **DateTime** вказаними значеннями року, місяця й дня, наприклад:

```
DateTime t1=DateTime(2015,11,9);           // 05.11.2018 0:00:00
DateTimeOffset t2=DateTime(2015,11,9);     // 05.11.2018 0:00:00 +02:00
```

DateTime(Int32, Int32, Int32, Int32, Int32, Int32) – ініціює новий екземпляр структури **DateTime** вказаними значеннями року, місяця, дня, години, хвилини та секунди, наприклад:

```
DateTime d=DateTime(2015,11,9,8,45,00);
Console.WriteLine(d.ToString());           // 05.11.2018 8:45:00
```

TimeSpan(Int32, Int32, Int32) – ініціалізує новий екземпляр структури **TimeSpan** вказаними значеннями годин, хвилин і секунд, наприклад:

```
TimeSpan t=TimeSpan(17,45,00);             // 17:45:00
```

Основні властивості **DateTime** та **DateTimeOffset**

Date – з повного значення «дата-час» повертає дату, а значення часу обнулює, наприклад:

```
DateTime d = DateTime::Parse("04.12.2018 17:45:00");
DateTime dateOnly = d.Date;
Console.WriteLine(dateOnly.ToString());     // 04.12.2018 00:00:00
Console.WriteLine(dateOnly.ToString("d"));  // 04.12.2018
```

Day – з повного значення «дата-час» повертає ціле значення дня місяця:

```
int day = d.Day; // 04.12.2018
Console.WriteLine(day); // day = 4
```

Month – зі значення «дата-час» повертає ціле значення місяця від 1 до 12.

Year – зі значення «дата-час» повертає ціле значення року.

Hour – зі значення «дата-час» повертає ціле значення годин від 0 до 23.

Minute – зі значення «дата-час» повертає ціле значення хвилин від 0 до 59.

DayOfWeek – повертає день тижня як одне зі значень діапазону Sunday – Saturday, які в разі переведення до цілих чисел, відповідають діапазону 0 – 6:

```
DateTime d = DateTime.Parse("05.11.2018 17:45:00");
Console.WriteLine(d.DayOfWeek); // Monday
```

Today – повертає поточну системну дату (без часу).

Now – повертає поточну системну дату і час.

```
DateTime t = DateTime.Today; // 05.11.2018 0:00:00
DateTime t = DateTime.Now; // 05.11.2018 10:52:36
```

Ticks – повертає ціле число (тип Int64) тактів, які складають зазначені дату і час. До речі, один такт – це 100 наносекунд або одна десятимільйонна секунди. Значення властивості **Ticks** для зазначеного екземпляра означає кількість 100-наносекундних інтервалів, які пройшли з 00:00:00 годин 1 січня 0001 року.

Основні методи

Add – додає до дати і часу (типу DateTime, DateTimeOffset або TimeSpan) значення типу TimeSpan і повертає новий об'єкт DateTime, наприклад:

```
DateTime d1 = DateTime.Now;
Console.WriteLine(" {0:dd.MM.yy dddd}", d1); // 05.11.18 понеділок
TimeSpan duration( 100, 0, 0, 0 ); // 100 днів
DateTime d2 = d1.Add( duration );
Console.WriteLine(" {0:dd.MM.yy dddd}", d2); // 13.12.19 середа

TimeSpan t = TimeSpan.Parse("15:45");
t = t.Add(t); // 15:45 + 15:45 = 31.5 година = 1.07:30 (один день і 7.5 годин)
```

AddDays – додає до дати і часу (типу DateTime або DateTimeOffset) зазначену кількість днів (дійсне число) і повертає новий об'єкт DateTime, наприклад:

```
DateTime d = DateTime.Parse("05.12.2018 17:45:00");
d = d.AddDays(2.5); // 08.12.2018 5:45
```

AddMonths – додає до дати і часу (типу DateTime або DateTimeOffset) зазначену кількість місяців (дійсне число) і повертає новий об'єкт DateTime.

AddYears – додає до дати і часу (типу DateTime або DateTimeOffset) зазначену кількість років (дійсне число) і повертає новий об'єкт DateTime.

AddHours – додає до дати і часу (типу DateTime або DateTimeOffset) зазначену кількість годин (дійсне число) і повертає новий об'єкт DateTime.

AddMinutes – додає до дати і часу (типу DateTime або DateTimeOffset) зазначену кількість хвилин (дійсне число) і повертає новий об'єкт DateTime.

Compare – порівнює дві дати (типу DateTime, DateTimeOffset або TimeSpan) і повертає ціле число (додатне, якщо перша дата більша, ніж друга, і від'ємне, якщо навпаки), наприклад:


```
int n = DateTime::Compare(d1,d2);
if (n < 0) Console::WriteLine("d1 раніш d2");
else
    if (n > 0) Console::WriteLine("d1 пізніше d2");
    else Console::WriteLine("d1 = d2");
```

DaysInMonth – повертає кількість днів у зазначеному місяці 28(29), 30 чи 31:

```
int n = DateTime::DaysInMonth(2000,2); // n = 29
```

Equals – визначає, чи є однаковими дві дати (типу `DateTimeOffset`, `DateTime` або `TimeSpan`). Існують різні форми цього методу з одним і двома аргументами, які є схожими, позаяк повертають логічне значення `true` чи `false`. Так, наступні дві команди по суті є ідентичними:

```
bool n = d1.Equals(d2);
bool n = DateTime::Equals(d1,d2);
```

Parse(String) – перетворює рядок на дату типу `DateTime`:

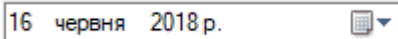
```
DateTime d = DateTime::Parse("25.12.18 17:45");
```

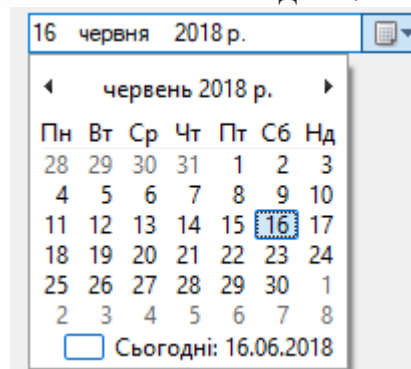
Subtract – віднімає значення аргументу – дати і часу (тип `DateTime`) або інтервалу часу (тип `TimeSpan`) і повертає нове значення:

```
DateTime d1, d2, d3, d4, d5; TimeSpan diff1, diff2;
d1 = DateTime( 2018, 6, 20, 22, 15, 0 ); // 20.06.2018 22:15
d2 = DateTime( 2018, 12, 20, 12, 2, 0 ); // 20.12.2018 12:02
d3 = DateTime( 2018, 12, 30, 18, 42, 0 ); // 30.12.2018 18:42
diff1 = d2.Subtract( d3 ); // diff1=d2-d3 = -10.06:40
d4 = d3.Subtract( diff1 ); // 10.01.2016 01:22
diff2 = d3 - d2; // diff2=d3-d2 = 10.06:40
d5 = d1 - diff2; // 10.06.2018 15:35
```

ToString() – перетворює значення дати і часу на рядок. Приклади використання цього методу з можливостями форматного виведення значень дати і часу розглядались на с. 391.

11.2.3 Елементи `DateTimePicker` та `MonthCalendar`

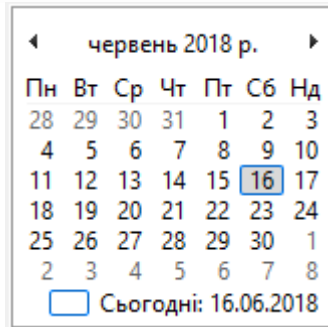
Елемент **`DateTimePicker`** Windows Forms дозволяє користувачеві вибрати значення дати і часу зі списку у вигляді календарика , при чому сам календарик розкривається при натисканні кнопки зі стрілкою, розташованої праворуч від значення поточної дати:



Зчитати вибране на `dateTimePicker` значення дати з метою подальшого програмного опрацювання можна такою командою:

```
DateTime d = dateTimePicker1->Value;
```

Елемент **MonthCalendar** надає користувачам зрозумілий графічний інтерфейс для перегляду і вибору одного чи навіть декількох значень дати:



Приклад 11.8. Ввести дату кінцевого терміну придатності деякого продукту та визначити, через скільки днів цей термін скінчиться, порівнявши його з поточною датою.

Текст програми:

```
using namespace System;
int main(array<System::String ^> ^args)
{ String^ dstr;
  Console::WriteLine("Ввести термін придатності");
  dstr = Console::ReadLine();
  DateTime dateTerm, dateNow;
  dateTerm = DateTime::Parse(dstr);           // Перетворити рядок на дату
  dateNow = DateTime::Now;                    // Визначити поточну дату
  if (dateTerm > dateNow)                     // Якщо кінцевий термін ще не настав,
  { TimeSpan days = dateTerm - dateNow;       // обчислити різницю дат
    Console::WriteLine ("Термін придатності скінчиться через " + days.Days
                        + " дня(ів)");
  } else Console::WriteLine("Продукт прострочений");
  Console::Read();
  return 0;
}
```

Результати виконання програми:

Ввести термін придатності 22.12.2014 Термін придатності скінчиться через 123 дня(ів)	Ввести термін придатності 12.05.2013 Продукт прострочений

Приклад 11.9. Відібрати з введеної інформації про абонентів (номер мобільного телефону, дата підключення (активації) номера, назву тарифу, вартість однієї хвилини розмов, прізвище та ім'я) дані про абонентів, які користуються мобільним зв'язком понад 10 років без змінення номера телефону.

Розв'язок. Умову відбору можна організувати в різний спосіб.

Наприклад, можна обчислити інтервал між нинішньою системною датою і датою зазначеною у рядку (друга лексема). Властивість `Days` визначає кількість днів періоду часу, заданого структурою `TimeSpan`:

```
DateTime d = DateTime::Parse(substrings[1]);
TimeSpan y = DateTime::Today - d; // обчислити різницю дат
```

```
if (y.Days/365.25 >= 10) . . .
```

Іншим способом є використання властивості Year структури DateTime, яка вибирає значення року зі значення дати:

```
DateTime d = Convert.ToDateTime(substrings[1]);
```

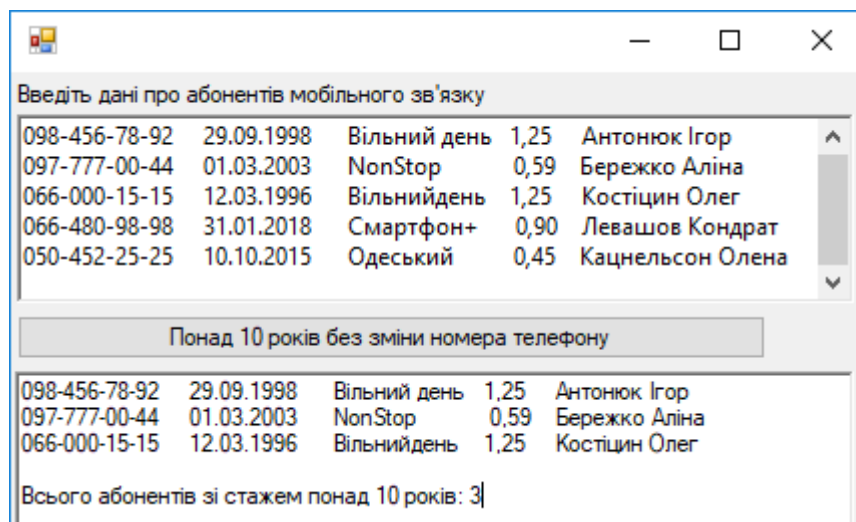
```
if (DateTime.Today.Year - d.Year >= 10) . . .
```

Наведені методи DateTime.Parse та Convert.ToDateTime є тотожними за дією і перетворюють рядок на значення типу DateTime.

Текст програми:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{ richTextBox2->Clear();
  int i, count=0; String ^s, ^copy; DateTime d;
  array<Char> ^separator = { ' ', '\t' };
  for (i=0; i<richTextBox1->Lines->Length; i++)
  { s=copy=richTextBox1->Lines[i];
    array<String> ^substrings = s->Split(separator,
                                          StringSplitOptions::RemoveEmptyEntries);
    d = DateTime::Parse(substrings[1]);
    if ( d.Year+10 <= DateTime::Today.Year )
    { richTextBox2->Text+=copy+"\n";
      count++;
    }
  }
  richTextBox2->Text += "\nВсього абонентів зі стажем понад 10 років: "
    + count.ToString();
}
```

Результат виконання програми:



Приклад 11.10. Створити текстовий файл з інформацією про дні народження друзів: прізвище, ім'я, дата народження. Переглянути файл. Визначити найстаршого з друзів, а також тих, хто святкуватиме день народження у наступному місяці.

Текст програми:

```
using namespace System;
using namespace System::IO;

void WriteToFile(String^ fname)    // Формування файлу з даними
{ String ^surn, ^name, ^birthd;
  do
  { Console::Write("Введіть прізвище ");    surn=Console::ReadLine();
    if (surn != "")
    { Console::Write("Введіть ім'я ");      name=Console::ReadLine();
      Console::Write("Введіть дату народження "); birthd=Console::ReadLine();
      File::AppendAllText(fname, surn+" \t" +name+" \t"+birthd+ "\n");
    }
  } while (surn != "");
}

void MaxAge(String^ fname)          // Пошук найстаршого з друзів
{ Console::WriteLine("\nНайстарший з друзів");
  String ^maxbirth, ^maxsurn, ^maxname, ^surn, ^name, ^birthd, ^str;
  DateTime maxB;
  array<Char> ^separator = { ' ', '\t' };
  StreamReader ^f = File::OpenText( fname );
  try
  { str = f->ReadLine();                // Зчитати перший рядок з файлу
    array<String^> ^substrings = str->Split(separator,
      StringSplitOptions::RemoveEmptyEntries); // Розбити рядок на слова
    maxsurn = substrings[0];
    maxname = substrings[1];
    maxbirth = substrings[2];
    maxB = DateTime::Parse(maxbirth);
    while (str = f->ReadLine())
    { array<String^> ^substrings = str->Split(separator,
      StringSplitOptions::RemoveEmptyEntries);
      surn = substrings[0];
      name = substrings[1];
      birthd = substrings[2];
      if (DateTime::Parse(birthd)<maxB) //Якщо дата народження менша за максимальну
      { maxbirth = birthd;    maxsurn = surn;
        maxname = name;      maxB = DateTime::Parse(maxbirth);}
    }
    Console::WriteLine(maxsurn+" "+maxname+" "+maxbirth );
  }
  finally
  { f->Close(); }
}

void NextMonth(String^ fname) // Іменинники наступного місяця
{ Console::WriteLine("\nДрузі, які святкуватимуть день народження у наступному місяці");
  Console::WriteLine("Поточний місяць - "+ (DateTime::Now).ToString(L"MMM"));
```

```

StreamReader ^f = File::OpenText( fname );
array<Char> ^separator = { ' ', '\t' };
try
{
    DateTime birthd; String ^scopy, ^str;
    while (f->Peek() > 0)
    {
        str = f->ReadLine();
        scopy = str; // Зробити копію рядка, а сам рядок поділити на слова
        array<String^> ^substrings = str->Split(separator,
            StringSplitOptions::RemoveEmptyEntries);
        birthd = DateTime::Parse(substrings[2]); // Третє "слово" рядка – дата народження
        if ((birthd.Month == 1 && DateTime::Now.Month == 12) ||
            (birthd.Month == DateTime::Now.Month+1)) Console::WriteLine(scopy);
    }
}
finally { f->Close(); }
}

int main(array<System::String ^> ^args)
{
    String^ fname = "birthd.rtf";
    Console::WriteLine("Введіть дані, які будуть записані у файл. Наприкінці двічі натисніть Enter.");
    WriteToFile (fname); // Формування файлу з даними
    Console::WriteLine("\n Вміст файлу:\n");
    Console::WriteLine(File::ReadAllText (fname));
    MaxAge (fname); // Пошук найстаршого з друзів
    NextMonth (fname); // Іменинники наступного місяця
    Console::ReadLine();
    return 0;
}

```

Результат виконання програми:

```

Введіть дані, які будуть дописані у файл. Наприкінці двічі натисніть Enter.
Введіть прізвище Бутур
Введіть ім'я Юрій
Введіть дату народження 05.08.1999
Введіть прізвище

Вміст файла:
Ткачук Олег 11.02.1998
Баранов Ігор 25.07.1997
Васильюк Ольга 21.01.1996
Петрова Ірина 02.12.1998
Харчук Іван 31.01.1999
Ященко Захар 01.01.1997
Коритко Олена 12.03.1997
Званчук Коля 11.09.1998
Гаража Юлія 04.05.1999
Бутур Юрій 05.08.1999

Найстарший з друзів
Васильюк Ольга 21.01.1996

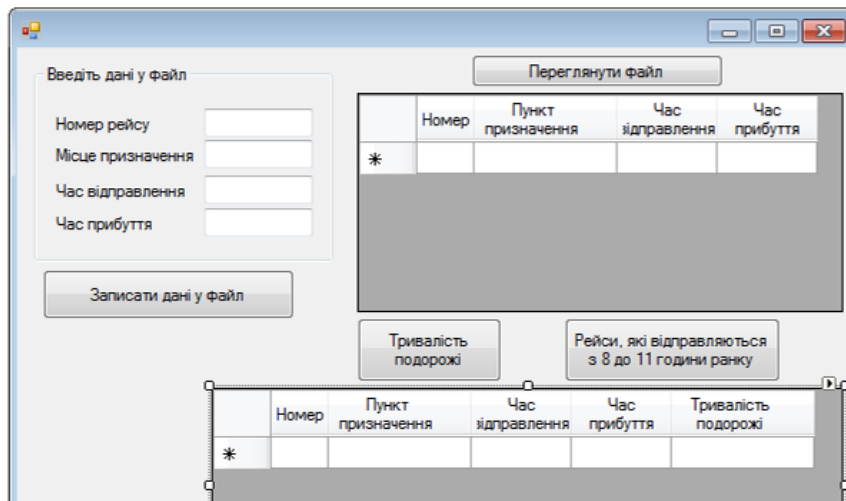
Друзі, які святкуватимуть день народження у наступному місяці
Поточний місяць – Грудень
Васильюк Ольга 21.01.1996
Харчук Іван 31.01.1999
Ященко Захар 01.01.1997

```

Приклад 11.11. Створити файл з даними про рух автобусів: номер рейсу, місце призначення, час відправлення, час прибуття. Обчислити для кожного з рейсів його тривалість. Знайти автобуси, які відправляються з 8 до 11 години ранку.

Розв'язок. При створенні проекту з формою при її налаштуванні доцільно розмістити на ній:

- для введення даних – чотири `textBox` з чотирма відповідними `label` для пояснювальних підписів. З метою угрупування ці вісім елементів можна розташувати на `groupBox1`, у властивості `Text` якого записати «Введіть дані у файл»;
- чотири кнопки `button`;
- для переглядання файлу – `dataGridView1` з чотирма стовпцями, для яких вписати відповідні заголовки стовпців і, можливо, задати їх ширину та ін.;
- для почергового виведення результатів завдань про тривалість подорожей і ранкових рейсів – елемент `dataGridView2`. Щоб не налаштовувати цей елемент "з нуля" доцільно просто скопіювати і вставити елемент `dataGridView1`. При цьому автоматично створиться `dataGridView2` з чотирма вже налаштованими стовпцями і залишиться долучити ще один (п'ятий) стовпець з заголовком "Тривалість подорожі". Оскільки цей п'ятий стовпець не буде потрібним при виведенні даних про ранкові рейси, будемо у програмі тимчасово його вилучати чи долучати.



Текст програми:

```
String^ fname; // Глобальна змінна для імені файлу
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{fname="Buses.rtf"; // Ім'я файлу
 dataGridview1->AllowUserToAddRows = false; //Заборона автоматичного створення рядка
 dataGridview2->AllowUserToAddRows = false;
}
// Кнопка "Записати дані у файл"
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{ // Записати у файл рядок з даними усіх чотирьох textBox-ів
 File::AppendAllText(fname, textBox1->Text+"\t " + textBox2->Text+"\t " +
                      textBox3->Text+"\t " + textBox4->Text+"\n");
 textBox1->Clear(); textBox2->Clear(); textBox3->Clear(); textBox4->Clear();
}
```



```
// Кнопка "Переглянути файл"
```

```
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e)
{
    dataGridView1->Rows->Clear();    // Очищення елемента
    if (File::Exists(fname))        // Якщо файл існує, виконати подальші дії
    { String ^str;
      array<Char>^ separator = { ' ', '\t' };    // Роздільники слів у рядку файлу
      StreamReader^ f = File::OpenText(fname);    // Відкрити файл для читання
      try
      { // Почергово читати рядки файлу, розділяти на слова
        // і виводити їх у клітинки dataGridView1
        while (f->Peek() > 0)
        { s = f->ReadLine() ;
          array<String>^ ^subs = str->Split(separator,
                                             StringSplitOptions::RemoveEmptyEntries);
          dataGridView1->Rows->Add(subs[0],subs[1],subs[2],subs[3]);
        } }
      finally
      { f->Close(); }
    } }
}
```

```
// Кнопка "Тривалість подорожі"
```

```
private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e)
{ dataGridView2->Rows->Clear();
  if(!dataGridView2->Columns->Contains("Column5")) // Якщо п'ятого стовпця немає,
    dataGridView2->Columns->Add("Column5", "Тривалість подорожі"); //додати його
  if (File::Exists(fname))        // Якщо файл існує, виконати подальші дії
  { String^ str;
    DateTime departTime, arrivTime;    TimeSpan vputi, t;
    array<Char>^ separator = { ' ', '\t' };
    StreamReader^ f = File::OpenText(fname);
    try
    while (f->Peek() > 0)
    { s = f->ReadLine() ;
      array<String>^ ^subs = str->Split(separator,
                                         StringSplitOptions::RemoveEmptyEntries);
      departTime = Convert::ToDateTime(subs[2]); // Перетворення рядка на дату
      arrivTime = DateTime::Parse(subs[3]);
      t = TimeSpan(24,0,0);    // t дорівнюватиме 24 годинам
      // Якщо час відправлення більше часу прибуття, додати до часу прибуття 24 години
      if (DateTime::Compare(departTime, arrivTime) > 0)
        vputi = arrivTime.Add(t) - departTime;
      else vputi = arrivTime - departTime;
      dataGridView2->Rows->Add(subs[0],subs[1],subs[2],subs[3],vputi);
    } }
    finally
    { f->Close(); }
  }
}
```



```
// Кнопка "Рейси, які відправляються з 8 до 11 години ранку"
private: System::Void button4_Click(System::Object^ sender, System::EventArgs^ e)
{ dataGridView2->Rows->Clear();
  if (Column5) dataGridView2->Columns->RemoveAt(4); //Якщо 5-ий стовпець є, видалити його
  if (File::Exists(fname))
  { String^ s;    array<Char>^ separator = { ' ', '\t' };
    StreamReader^ f = File::OpenText(fname);
    DateTime departTime;
    try
    { while (s = f->ReadLine())
      { array<String>^ ^subs = s->Split(separator,
                                         StringSplitOptions::RemoveEmptyEntries);

        departTime = DateTime::Parse(subs[2]);
        // Якщо відправлення автобусу між 8 та 11 годинами ранку
        if (departTime >= DateTime::Parse("08:00:00") &&
            departTime <= DateTime::Parse("11:00:00"))
          dataGridView2->Rows->Add(subs[0], subs[1], subs[2], subs[3]);
      }
    }
    finally { f->Close(); }
  }
}
```

Результати виконання програми:

The application interface includes the following components:

- Input Fields:**
 - Номер рейсу (Route Number)
 - Місце призначення (Destination)
 - Час відправлення (Departure Time)
 - Час прибуття (Arrival Time)
- Buttons:**
 - Введіть дані у файл (Enter data in file)
 - Записати дані у файл (Save data to file)
 - Переглянути файл (View file)
 - Тривалість подорожі (Travel duration)
 - Рейси, які відправляються з 8 до 11 години ранку (Routes departing from 8 to 11 AM)
- Tables:**
 - Table 1 (Main):**

Номер	Пункт призначення	Час відправлення	Час прибуття
123	Львів	15:45	23:55
125	Дніпропетровськ	20:00	9:20
7	Київ	10:05	17:30
8	Одеса	10:45	19:05
34	Полтава	8:45	6:45
 - Table 2 (Filtered):**

Номер	Пункт призначення	Час відправлення	Час прибуття	Тривалість подорожі
123	Львів	15:45	23:55	08:10:00
125	Дніпропетровськ	20:00	9:20	13:20:00
7	Київ	10:05	17:30	07:25:00
8	Одеса	10:45	19:05	08:20:00
 - Table 3 (Detailed View):**

Номер	Пункт призначення	Час відправлення	Час прибуття
7	Київ	10:05	17:30
8	Одеса	10:45	19:05
34	Полтава	8:45	6:45

Питання та завдання для самоконтролю

- 1) Які заголовні файли слід долучити до консольної програми для використання функцій опрацювання дати і часу?
- 2) Охарактеризувати тип `time_t`.
- 3) У чому специфіка структури `tm` з бібліотеки `time.h`?
- 4) Яке призначення функції `time` з бібліотеки `time.h`?
- 5) Яке значення буде результатом функції `difftime`?
- 6) Охарактеризувати структуру `DateTime`.
- 7) У чому специфіка структури `DateTimeOffset`?
- 8) У чому схожість і відмінність структур `DateTime` і `DateTimeOffset`?
- 9) Охарактеризувати структуру `TimeSpan`.
- 10) Чим тип `DateTime` відрізняється від типу `TimeSpan`?
- 11) Яка властивість структури `DateTime` дозволяє визначити поточну системну дату (без часу)?
- 12) Як визначити і системну дату і час?
- 13) Як вивести тільки час (без дати)?
- 14) Як вивести значення з повною назвою дня тижня дати?
- 15) Записати команди, які дозволять збільшити (чи зменшити) дату:
а) на один день; б) на один тиждень; в) на один місяць?
- 16) Як перетворити рядок на значення типу `DateTime`?
- 17) Записати приклад перетворення рядка на значення типу `DateTime`?
- 18) У чому схожість і відмінність методів `DateTime::Parse` та `Convert::ToDateTime`?
- 19) Які методи дозволяють перетворити значення типу `DateTime` на рядок?
- 20) Який метод дозволяє порівняти дві дати? Що означатиме від'ємний результат цього методу?
- 21) Навести приклад команди для порівняння двох дат.
- 22) В який спосіб можна за датою визначити номер місяця?
- 23) Як за датою визначити номер дня тижня?
- 24) Записати команду, яка визначатиме значення години заданого часу.
- 25) Записати команду, яка визначатиме значення дня заданої дати.
- 26) Який метод перетворить рік, місяць і день цілого типу на значення типу `DateTime`.
- 27) Яке призначення методу `AddDays`? Навести приклад.
- 28) Яке призначення методу `Subtract`? Навести приклад.
- 29) У чому схожість і відмінність властивостей `Today` та `Now`?
- 30) Назвати спеціальні елементи керування Visual C++ для роботи з датою і часом.

РОЗДІЛ 12

ФАЙЛИ

12.1 Загальні відомості про файли

Файлом є іменований блок даних, який зберігається на носії інформації. Файли мають *імена*. Вони організовані в деревоподібну структуру з *каталогів* (папок, тек) і простих файлів.

Часто використовується така метафора: якщо уявляти собі файли як книжки (лише зчитування) і блокноти (зчитування й записування), які стоять на полиці, то відкриття **файлу** – це вибір книжки чи блокнота за заголовком на його обкладинці й відкриття обкладинки (на першій сторінці). Після відкриття можна читати, дописувати, викреслювати і правити записи, перегортати книжку. Сторінки можна зіставити з блоками **файлу**, а полицю з книжками – з каталогом.

Для доступу до даних файлу з програми в ній слід прописати функцію відкриття чи створення цього файлу, тим самим встановити зв'язок між ім'ям файлу і певною файловою змінною у програмі. Доступ до даних файлу відбувається з так званої позиції зчитування-записування, яка автоматично просувається при операціях зчитування-записування, тобто файл є видимим послідовно. Існують, щоправда, функції для довільного змінювання цієї позиції.

C++ надає засоби опрацювання двох типів файлів: *текстових* і *бінарних*. Ці типи визначають спосіб роботи з даними у файлі та не залежать від розширення або системного типу файлу.

Текстові файли – це файли, робота над якими виконується по рядках – зазвичай за один крок зчитується або записується один рядок. Такі файли розглядаються як сукупність символьних рядків змінної довжини. Кожен рядок завершується керувальною послідовністю '`\n`', а розділювачами слів та чисел у рядку є пробіли й символи табуляції. Оскільки вся інформація текстового файлу є символьною, програмне опрацювання такого файлу полягає в читанні рядків, виокремлюванні з рядка слів і, за потреби, перетворюванні цифрових символьних послідовностей на числа відповідними функціями перетворювання. Створювати і редагувати текстові файли можна не лише в програмі, а і в якому завгодно текстовому редакторі, наприклад, Word, WordPad чи Блокноті.

Під час роботи з **бінарними (двійковими) файлами** зчитують та записують дані не по рядках, а блоками даних встановленого розміру. Такі файли зберігають дані у тому самому форматі, в якому вони були оголошені, а їхній вигляд є такий самий, як і в пам'яті комп'ютера. І тому відпадає потреба у використанні розділювачів: пробілів, керувальних послідовностей, а отже, розмір використовуваної пам'яті порівняно з текстовими файлами з аналогічною інформацією є значно меншим. Окрім того, немає потреби у застосуванні функцій перетворювання числових даних. Але кожне опрацювання даних бінарних файлів можливе лише за наявності програми, якій має бути відомо, що саме і в якій послідовності зберігається у цьому файлі.

Незалежно від типу файлу алгоритм його опрацювання є таким:

- 1) відкрити файл;
- 2) виконати дії над даними у файлі;
- 3) закрити файл.

Кожний файл має ім'я (наприклад, `myfile.txt`). Файл у відповідній програмі опрацювання пов'язується зі змінною. Будь-які дії, які виконуються над файловою змінною, автоматично реалізуються над фізичним файлом на диску.

Послідовно розглянемо засоби створення, записування, зчитування й опрацювання файлів обох типів.

Робота з файлами у Visual C++ може виконуватися кількома різними засобами:

- ✓ робота з файлами як з потоками у стилі C;
- ✓ робота з файлами як з потоками у стилі C++;
- ✓ опрацювання файлів засобами простору імен `System::IO` платформи .NET Framework;
- ✓ методами елементів керування Visual C++.

Спочатку розглянемо можливості для створення й опрацювання текстових файлів.

12.2 Текстові файли

12.2.1 Робота з текстовими файлами у стилі C

У мові C *файл* розглядається як потік послідовності байтів. Вся робота з файлом виконується через файлову змінну – вказівник на структуру типу `FILE`, означену в стандартній бібліотеці `stdio.h`. Більше того, у цій бібліотеці описані усі базові функції для роботи з файлами. Тому, якщо в програмі планується робота з файлами, цей заголовний файл слід долучити:

```
#include <stdio.h>
```

Оголошення файлової змінної – вказівника потоку може мати вигляд:

```
FILE *f;
```

Функція `fopen()` для відкривання файлу має такий синтаксис:

```
FILE *fopen(const char *filename, const char *mode);
```

Перший параметр `filename` визначає ім'я фізичного файлу, який відкривається і пов'язується з вказівником `f`. Другий параметр `mode` задає режим відкривання файлу, тобто визначає, які дії будуть доступні для виконання з даними у відкритому файлі (табл. 12.1).

До зазначених у табл. 12.1 специфікаторів наприкінці чи перед знаком "+" може дописуватись символ чи то "t" – для текстових файлів, чи "b" – для бінарних (двійкових) файлів.

Функція `fopen()` повертає вказівник на об'єкт, який керує потоком, тобто адресу пам'яті початку потоку. Наприклад, створити файл з ім'ям `Prim.txt` можна так:

```
f = fopen("Prim.txt", "wt");
```

Специфікатори режиму відкривання файлів

Параметр	Опис
r	відкрити файл лише для зчитування даних
r+	відкрити файл для зчитування й записування даних
a	відкрити чи створити файл для записування даних у кінець файлу
a+	відкрити чи створити файл для зчитування чи то записування даних у кінець файлу
w	створити файл для записування даних
w+	створити файл для зчитування і записування даних

Якщо файл відкрити не вдалося, `fopen()` повертає нульовий вказівник `NULL`. Для уникання помилок після відкриття файлу слід перевірити, чи насправді файл відкрився:

```
if (f == NULL) { cout<<"Файл не вдалось відкрити"; return; }
```

Припинити роботу з файлом можна за допомогою функції

```
fclose(FILE *f).
```

Ця функція закриває файл, на який посиляється параметр функції, зберігаючи змінення.

Наведемо приклад відкривання текстового файлу для зчитування.

```
FILE *f;
// Перевіряємо, чи повертає ця функція нульовий вказівник
if ((f=fopen("t.txt", "rt"))==NULL)
{ cout<<"Неможливо відкрити файл"; return; }
. . . // Сюди вставляються команди зчитування з файлу
fclose(f); // Закриття файлу
```

З текстового файлу можна читати цілі рядки й окремі символи. Слід звернути увагу на те, що тип `FILE*` перейшов в C++ зі стандартного C, і такий файл «розуміє» лише C-рядки, тобто рядки типу `char*`.

Зчитування рядка з файлу здійснюється функцією `fgets()`:

```
char *fgets(char *s, int m, FILE *stream);
```

де `s` – перший параметр – рядок типу `char*`;

`m` – кількість читаних символів (байтів);

`stream` – вказівник на потік даних файлу.

Перевірка кінця файлу здійснюється функцією `feof()`:

```
int feof (FILE *stream);
```

Розглянемо детальніше використання цих функцій:

```
char s[50];
do { fgets(s,50,f);
    cout<<s;
} while (!feof(f));
```

Даний приклад ілюструє зчитування даних з файлу порядково за допомогою функції `fgets()` і виведення рядків на екран. Перший параметр функції

`fgets()` – це C-рядок, в який читається черговий рядок текстового файлу. Зчитування рядка відбувається до появи символу кінця рядка `'\n'` або ж припиняється, коли прочитано $m-1$ символ. У нашому прикладі $m=50$. Отже, якщо файл містить рядок, який має понад 50 символів, то буде прочитано лише перші 49 символів. При цьому поточна позиція файлу залишиться в тому самому рядку й за подальшого використання функції `fgets()` читатиметься залишок рядка. Третій параметр зазначає потік, з якого здійснюється зчитування.

У даному прикладі при зчитуванні всіх даних з файлу `f` використовується функція `feof(f)`, яка перевіряє, чи не прочитано символ кінця файлу. Після зчитування цього символу функція `feof(f)` поверне ненульове значення – і цикл перерветься.

До речі, інколи використання функції `feof()` часто призводить до появи дублікату останнього рядка файлу, тому рекомендовано використовувати в якості умови циклу функцію зчитування даних, наприклад `fgets()`, або контролювати добігання кінця файлу, тобто перевіряти, чи прочитано всі записані у файлі символи (нагадаємо, що 1 символ – 1 байт). Наведений приклад зчитування рядків з файлу можна тепер записати в такий спосіб:

```
while (fgets(s,50,f))
{ if (strlen(s)-1=='\n') s[strlen(s)-1] = '\0';
  puts(s);
}
```

Зчитування форматованих даних можна також здійснювати за допомогою функції `fscanf()`:

```
int fscanf(FILE *stream, const char *format[, address.]);
```

Наведемо приклад використання цієї функції:

```
float r;
while ( fscanf(f,"%e",&r)>0 ) cout<<r;
```

У даному прикладі здійснюється зчитування даних з файлу, який містить дійсні числа. За розділювачі поміж числами вважаються пробіли. Перший параметр `f` функції `fscanf()` визначає файл, з якого відбувається зчитування. Другий параметр задає формат рядка аргументів, заданих їхніми адресами. Функція `fscanf()` є цілого типу і повертає, як своє вихідне значення, кількість прочитаних елементів. При форматованому читанні часто можуть виникати помилки через невідповідність форматів чи то кінець файлу. Для уникання таких помилок доцільно організовувати перевірки кількості прочитаних функцією `fscanf()` елементів. У наведеному прикладі формат `"%e"` визначає дійсне число з рухомою крапкою. Прочитане дійсне число з файлу зберігається за адресою змінної `r` типу `float`.

Рядок форматування параметра `format` будується з послідовності символів-специфікаторів типів читаних даних, найпоширеніші з яких наведено у табл. 12.2.

Записувати дані у текстовий файл можна за допомогою функції:

```
int fputs(const char *s, FILE *stream);
```

де `s` – рядок типу `char`;

`stream` – файловий потік.

Специфікатори параметра format

Символ	Значення, що вводиться	Тип аргументу
i, l	десятькове, вісімкове чи шістнадцяткове ціле число	int, long
d, D	десятькове ціле число	int, long
e, E	дійсне число з рухомою крапкою	float
f, F	дійсне число з фіксованою крапкою	float
s	рядок символів	char s[]
c	символ	char

Для записування даних до файлу слід відкрити файл у форматі записування даних у кінець файлу, за потреби перетворити рядок на тип `char` і записати дані до файлу. Послідовність процесу показано у фрагменті програми:

```
FILE *f;           // Оголошення файлової змінної
f=fopen("a.rtf", "at+");
if (f==0) { cout<<"Не вдається створити файл!"; return 0; }
char s[40];
gets(s);           // Введення рядка s
// Долучання до рядка символу [Enter], інакше у файлі
strcat(s, "\n"); // все буде записано одним рядком
fputs(s, f);       // Записування рядка s у файл
fclose(f);
```

Записування у текстовий файл можна здійснити також за допомогою функції `fprintf()`:

```
int fprintf(FILE *stream, const char *format [, argument]);
```

Ця функція є подібна до функції `fscanf()`, але має ширші можливості побудови рядка форматування, наприклад:

```
char s[20];
strcpy(s, "Іванов");
int year=1985;
fprintf(F, "ХАРАКТЕРИСТИКА\nспівробітник %s, %i р.н.\n", &s, year);
```

У результаті виконання цих команд до файлу буде записано таке:

```
ХАРАКТЕРИСТИКА
співробітник Іванов, 1985 р.н.
```

Як бінарні, так і текстові файли дозволяють переміщувати поточну позицію зчитування-записування. Для визначення поточної позиції файлу, яка автоматично зміщується на кількість опрацьованих байтів, використовується функція `ftell()`:

```
long int ftell(FILE *stream);
```

А змінити поточну позицію файлу можна за допомогою функції `fseek()`:

```
int fseek(FILE *stream, long offset, int whence);
```

Ця функція задає зсув на кількість байтів `offset` щодо точки відліку, яка задається параметром `whence`. Параметр `whence` може набувати таких значень:

Константа	whence	Точка відліку
SEEK_SET	0	початок файлу
SEEK_CUR	1	поточна позиція
SEEK_END	2	кінець файлу

Наприклад, для переміщення поточної позиції на початок файлу можна скористатися функцією

```
fseek(f, 0, SEEK_SET);
```

або

```
fseek(f, 0, 0);
```

За допомогою функцій `ftell()` та `fseek()` можна визначити сумарний обсяг пам'яті у байтах, який займає файл. Для цього є достатньо переміститися у кінець файлу:

```
fseek(f, 0, SEEK_END);
```

```
int d = ftell(f);
```

Отже, послідовність роботи з файлом при записуванні даних є така:

1) Відкрити чи то створити файл `fname.txt` для записування (або для зчитування й записування) у кінець файлу:

```
f=fopen("fname.txt", "at+");
```

2) Записати дані у файл `f` однією з команд:

```
fputs(s, f); // Записування C-рядка s
```

```
// Форматоване записування C-рядка nazva, дійсного числа price і цілого kol
```

```
fprintf(f, "%s %6.2f %i\n", nazva, price, kol);
```

3) Закрити файл:

```
fclose(f);
```

Для зчитування даних з файлу треба виконати таку послідовність дій:

1) Відкрити файл для зчитування

```
f=fopen("fname.txt", "rt+");
```

2) Здійснити зчитування даних з файлу однією з команд:

```
fgets(s, 80, f); // Зчитування C-рядка s довжиною у 80 символів
```

```
// Форматоване зчитування за адресами відповідних змінних
```

```
fscanf(f, "%s %f %i\n", &nazva, &price, &kol);
```

3) Для зчитування послідовно всіх даних з файлу треба організувати цикл з умовою, яка перевіряє досягання кінця файлу, використовуючи чи то функцію `feof(f)`, яка є ідентична до `true` при досяганні кінця файлу, чи функцію зчитування даних, яка дає нульовий результат за досягання кінця файлу. Можна використовувати інформацію про сумарний розмір файлу і контролювати досягання кінця файлу функцією `ftell(f)`.

4) Закрити файл: `fclose(f)`;

У прикладах, що наводяться нижче, всі ці можливості послідовно подано у програмах.

Приклад 12.1. Створити в корневій папці диску D за допомогою програми *Блокнот* текстовий файл з ім'ям `myfile.txt` і ввести до нього кілька рядків. На-

писати програму, яка виводить зміст цього файлу на екран і обчислює кількість рядків у файлі та їхню середню довжину.

Текст програмного коду:

```
#include <iostream>
#include <stdio.h>
using namespace std;
int main()
{
    setlocale(0, ".1251");
    char* name="d:\\myfile.txt";
    FILE* f;
    char s[100];
    int sum=0, kol=0;
    f=fopen(name, "rt"); // Відкрити файл для читання як текстовий.
    if(f==NULL)
    {cout<<"Неможливо відкрити файл\n";
      system("pause"); return 0;
    }
    cout<<"\nПерегляд файлу"<<endl;
    while (fgets(s, 100, f)) // Зчитувати рядки з файлу,
    { // допоки вони не закінчаться (до кінця файлу).
      // Видалити останній символ (інакше після кожного рядка буде виведено порожній).
      s[strlen(s)-1]='\0';
      puts(s); // Вивести рядок на екран.
      // Додати довжину цього рядка до загальної довжини файлу.
      sum+=strlen(s)-1;
      kol++; // Збільшити кількість рядків.
    }
    fclose(f);
    cout<<"\nКількість рядків у файлі="<<kol<<endl;
    // Якщо у файлі є рядки, обчислити та вивести середню довжину рядків
    if (kol)
      cout<<"\nСередня довжина рядків="<<sum/kol<<endl;
    system("pause");
    return 0;
}
```

Результати виконання програми:

Перегляд файлу

У даному прикладі здійснюється зчитування даних з файлу.

За розділювачі поміж числами вважаються пробіли.

Перший параметр *f* функції *fscanf()* визначає файл.

Другий параметр задає формат рядка аргументів, заданих їхніми адресами.

Кількість рядків у файлі=5

Середня довжина рядків=44

Приклад 12.2. Створити текстовий файл (кінець введення – порожній рядок). Вивести спочатку увесь вміст файлу на екран, а тоді рядки довжиною понад 20 символів.

Текст програмного коду:

```
#include <iostream>
#include <conio.h>
#include <stdio.h>
using namespace std;
void create_file(char* name) // Функція створення файлу
{ FILE* f;
  char s[100];
  f=fopen(name, "wt"); // Відкрити файл для створення як текстовий.
  if (f==NULL) // Перевірити, чи відкрився файл.
  { cout<<"Неможливо створити файл\n"; return;}
  cout<<"Введіть рядки"<<endl;
  do // Цикл, поки не введено порожній рядок,
  { gets_s(s,100); // ввести рядок із клавіатури,
    fputs(s, f); // записати його до файлу
    fputs("\n", f); // і перейти на новий рядок.
  } while (strcmp(s,""));
  fclose(f); // Закрити і зберегти файл.
}
void view_file(char* name) // Функція переглядання файлу
{ FILE* f;
  char s[100];
  f=fopen(name, "rt"); // Відкрити файл для читання як текстовий.
  if(f==NULL) {cout<<"Неможливо відкрити файл\n"; return;}
  cout<<"\nПерегляд файлу"<<endl;
  while (fgets(s, 100, f))
  { s[strlen(s)-1]='\0';
    puts(s);
  }
  fclose(f);
}
// Функція виведення рядків довжиною понад 20 символів.
void strings20(char* name)
{ FILE* f; char s[100];
  f=fopen(name, "rt");
  if(f==NULL) {cout<<"Cannot open file\n"; return;}
  cout<<"\nРядки довжиною понад 20 символів:"<<endl;
  while (fgets(s, 100, f))
  { s[strlen(s)-1]='\0';
    if (strlen(s)>20) puts(s);
  }
  fclose(f);
}
```

```
int main()
{ char* name="myfile.txt";    // Ім'я фізичного файлу в папці проекту.
  create_file(name);
  view_file(name);
  strings20(name);
  system("pause");
  return 0;
}
```

Результати виконання програми:

Введіть рядки

There are a large number of functions
to handle file I/O (Input Output) in C
When dealing with files, there are two types of files:
1. Text files
2. Binary files

Перегляд файлу

There are a large number of functions
to handle file I/O (Input Output) in C
When dealing with files, there are two types of files:
1. Text files
2. Binary files

Рядки довжиною понад 20 символів:

There are a large number of functions
to handle file I/O (Input Output) in C
When dealing with files, there are two types of files:

Приклад 12.3. Створити файл з рядків (кінець введення – порожній рядок).
Визначити кількість рядків, які містять цифри.

Розв'язок. Для створення та переглядання файлу скористаймося функціями `create_file` і `view_file` з прикладу 12.2. За потреби додамо команди змінення локалізації.

Текст програмного коду:

// Функція виведення рядків з цифрами та обчислення їх кількості.

```
int digits(char* name)
{ FILE* f;
  char s[100];
  int k=0;
  f=fopen(name, "rt");
  if (f==NULL) {cout<<"Cannot open file\n"; return 0;}
  setlocale(0, ".1251");
  cout<<"\nРядки, які містять цифри:"<<endl;
  setlocale(0, ".ОСР");
  while (fgets(s, 100, f))
  { s[strlen(s)-1]='\0';
    for (int i=0; i<strlen(s); i++)
```

```
        if (s[i]>='0' && s[i]<='9') { k++; puts(s); break;}
    }
    fclose(f);
    return k;
}
int main()
{ setlocale(0, ".1251");
  char* name="myfile.txt";
  create_file(name);
  view_file(name);
  int kol=digits(name);
  setlocale(0, ".1251");
  cout<<"\nКількість рядків з цифрами="<<kol<<endl;
  system("pause");
  return 0;
}
```

Результати виконання програми:

Введіть рядки

C++ - мова програмування високого рівня.

Розроблена Б'ярном Страуструпом

1979 року та початково отримала назву <Сі з класами>.

Згодом Страуструп перейменував мову на C++ у 1983 р.

Базується на мові С. Вперше описана стандартом

ISO/IEC 14882:1998, найбільш актуальним же є стандарт ISO/IEC 14882:2014

Перегляд файлу

C++ - мова програмування високого рівня.

Розроблена Б'ярном Страуструпом

1979 року та початково отримала назву <Сі з класами>.

Згодом Страуструп перейменував мову на C++ у 1983 р.

Базується на мові С. Вперше описана стандартом

ISO/IEC 14882:1998, найбільш актуальним же є стандарт ISO/IEC 14882:2014

Рядки, які містять цифри:

1979 року та початково отримала назву <Сі з класами>.

Згодом Страуструп перейменував мову на C++ у 1983 р.

ISO/IEC 14882:1998, найбільш актуальним же є стандарт ISO/IEC 14882:2014

Кількість рядків з цифрами=4

Приклад 12.4. Створити файл з рядків (кінець введення – порожній рядок). Вивест слова, які починаються і закінчуються на один і той самий символ.

Розв'язок. Для створення та переглядання файлу скористаймося функціями `create_file` і `view_file` з прикладу 12.2. За потреби додамо команди змінення локалізації.

Текст програмного коду:

```
// Функція виведення слів, які починаються і закінчуються
// на один і той самий символ.
void words(char* name)
{ FILE* f;
  char s[100], *t;
  f=fopen(name, "rt");
  if (f==NULL) {cout<<"Cannot open file\n"; return;}
  setlocale(0, ".1251");
  cout<<"\nСлова, які починаються та закінчуються на один і той самий
        СИМВОЛ:"<<endl;
  setlocale(0, ".OCP");
  while (fgets(s, 100, f))
  { s[strlen(s)-1]='\0';
    t=strtok(s, " .,;?!-");
    while (t != NULL)
    { if (t[0]==t[strlen(t)-1]) puts(t);
      t = strtok(NULL, " .,;?!-");
    }
  }
  fclose(f);
}
int main()
{
  setlocale(0, ".1251");
  char* name="myfile.txt";
  create_file(name);
  view_file(name);
  words(name);
  system("pause");
  return 0;
}
```

Результати виконання програми:

Введіть рядки

сос слон кіт

кошеня лелека апельсин

боб поле сніг піп

Перегляд файлу

сос слон кіт

кошеня лелека апельсин

боб поле сніг піп

Слова, які починаються та закінчуються на один і той самий символ:

сос

боб

піп

Приклад 12.5. Ввести відомості про товари у текстовий файл: назву, ціну товару та кількість товару на складі. Прочитати дані з файлу і віднайти відомості про товари, кількість яких є менше за 30. Визначити назву файлу.

Текст програми:

```
#include <stdio.h>
#include <iostream>
using namespace std;
int main( )
{ system("color F0");
  FILE *f;           // Оголосити файлову змінну
  char s[]="proba.txt"; // і визначити імені файлу.
  f = fopen(s,"rt+"); // Відкрити файл для зчитування і записування.
  if (f==NULL) f=fopen(s,"wt+");// якщо файл не було відкрито, створити його
  if (f==NULL) { cout<<"Не вдається відкрити файл!"; return 0; }
  fseek (f, 0, SEEK_END); // Перейти у кінець файлу
  char nazva[15]; float price; int kol;
  char yn;           // Ознака бажання дописувати нові рядки у файл.
rep:
  setlocale(0, ".1251");
  cout<<"Будете вводити рядки для записування у файл? у/н"<<endl;
  cin>>yn;
  if (yn!='y') goto peregljad;
  cout<<"Введіть рядок для записування у текстовий файл у форматі:
      \n назва товару, ціна, кількість на складі"<<endl;
  setlocale(0, ".OCP");
  scanf("%s %tf %ti",&nazva,&price,&kol);
  fprintf(f,"%s %t6.2f %ti\n", nazva, price, kol); // Дописати у файл.
  goto rep;
peregljad:
  fseek(f, 0, 0);
  setlocale(0, ".1251"); cout<<"\nВміст файлу:\n";
  setlocale(0, ".OCP");
  while (!feof(f)) // Контроль досягання кінця файлу.
  { // Зчитати рядок з файлу.
    fscanf(f,"%s %tf %ti\n",&nazva,&price,&kol);
    for (int i=strlen(nazva);i<=15; i++)strcat(nazva," ");
    printf("%s %t6.2f %ti\n",nazva,price,kol); // Вивести дані за форматом.
  }
  // Визначити розмір файлу.
  fseek(f, 0, SEEK_END);
  int len = ftell(f);
  setlocale(0, ".1251");
  cout<<"Розмір файлу - "<< len << "\n\n";
  fseek(f, 0, 0); // Перейти на початок файлу.
  setlocale(0, ".OCP");
  int n=0, sumn=0;
```



```

while (!feof(f))
{ fscanf(f,"%s \t%f \t%i\n",&nazva,&price,&kol); // Зчитати рядок з файлу
  if (kol<30)
  { n++; sumn += kol;
    for (int i=strlen(nazva); i<=15; i++) strcat(nazva," ");
    printf("%s \t%6.2f \t%i\n", nazva, price, kol);
  } }
setlocale(0, ".1251");
cout<<"\nВсього товарів, кількість яких є менше за 30, - "<< n
    <<" , їхня кількість - "<< sumn << endl;
fclose(f); // Закрити файл
system("pause");
return 0;
}

```

Результати роботи програми:

```

Будете вводити рядки для записування у файл? у/п
у
Введіть рядок для записування у текстовий файл у форматі:
назва товару, ціна, кількість на складі
Календар      45,60   40
Будете вводити рядки для записування у файл? у/п
п

Вміст файлу:
Пензлики      12,56   70
Кнопки        15,60   20
Маркер        7,84    67
Пенал         70,99   15
Зошит         17,45   50
Клей          23,10   10
Календар      45,60   40
Розмір файлу - 138

Кнопки        15,60   20
Пенал         70,99   15
Клей          23,10   10

Всього товарів, кількість яких є менше за 30, - 3, їхня кількість - 45

```

Введення даних до файлу з числового масиву

Для записування елементів числового масиву у файл існує дві можливості.

1) Перетворити елементи масиву на С-рядки, які записати у файл за допомогою функції `fputs()`. Наприклад, записування у файл цілочисельного масиву з 10-ти елементів:

```

char s[80];
for (int i=0; i<10; i++)
{ itoa(a[i],s,10); // Перетворити елемент на С-рядок,
  strcat(s, "\n"); // додати символ переведення на новий рядок
  fputs(s, f);     // і записати елемент у файл
}

```

У цьому прикладі кожен елемент перетворюється на рядок окремо від інших за допомогою функції `atoi()` (докладніше про цю функції див. у підрозд. 7.2) і записується у файл. В результаті всі числа масиву буде записано у файл у стовпчик. За потреби записувати елементи в один рядок через пробіл слід змінити тіло циклу в такий спосіб:

```
for (int i=0; i<10; i++)
{ itoa(a[i], s, 10); // Перетворити елемент на C-рядок,
  strcat(s, " ");    // долучити пробіл після чергового числа
  fputs(s, f);       // і записати число з пробілом у файл
}
fputs("\n", f);
```

Якщо кількість елементів є не надто велика, можна спочатку створити рядок, в який записати через пробіл усі елементи, а тоді записати цей рядок у файл:

```
char s[80]="", s1[10];
for (int i=0; i<10; i++) // До рядка долучити черговий елемент
{ itoa(a[i], s1, 10); strcat(s, s1); } // у символній формі
strcat(s, "\n");
fputs(s, f);
```

2) Записати елемент до файлу за допомогою функції форматowanego введення даних `fprintf()`:

```
for (int i=0; i<10; i++)
  fprintf(ft, "%6.2f\n", a[i]);
```

Також матрицю до файлу можна записати у такі способи.

1) Кожен елемент матриці перетворити до C-рядка і записати у файл окремо, після завершення кожного рядка матриці приєднати символ [Enter]:

```
for (int i=0; i<10; i++)
{ for (int j=0; j<10; j++)
  { itoa(a[i][j],s,10); // Перетворити число на C-рядок,
    strcat(s, "\t");    // долучити табуляцію після чергового числа
    fputs(s,f);        // і записати число з пробілом у файл
  }
  fputs("\n",f);       // Записати символ [Enter]
}
```

2) Сформувати C-рядок з елементів кожного рядка матриці й записати його у файл:

```
char s[80]="";
for (int i=0; i<10; i++)
{ for (int j=0; j<10; j++)
  { itoa(a[i][j],s,10); // Перетворити елемент на C-рядок,
    strcat(s, "\t");    // долучити табуляцію після чергового числа
  }
  strcat(s, "\n");     // і символ символ [Enter] у кінець рядка
  fputs(s, f);        // Записати рядок матриці у файл
}
```

3) Кожен елемент матриці записати у файл у числовій формі за допомогою функції `fprintf()`:

```

    for (int i=0; i<10; i++)
    { for (int j=0; j<10; j++)
        fprintf(ft, "%6.2f\n", a[i][j]); // Записати елемент
        fputs("\n", f); // i символ переведення курсора
    }

```

Приклад 12.6. Заповнити матрицю 5×5 випадковими числами і записати цю матрицю у текстовий файл.

Текст програми:

```

#include <iostream>
#include <time.h>
using namespace std;
int main()
{ system("color F0");
  int i,j,a[5][5]; char s[100];
  FILE* f;   time_t t;
  srand((unsigned) time(&t));
  for (i=0; i<5; i++)
  { for (j=0; j<5; j++)
      { a[i][j]=80-rand()%100;
        cout << a[i][j] << "t";
      } cout << endl;
  }
  if ((f=fopen("my_text.txt", "wt"))==0)
  { cout<<"Файл не вдалось відкрити "; cin.get(); return 0; }
  for (i=0; i<5; i++)
  { for (j=0; j<5; j++)
      { itoa(a[i][j],s,10); // Перетворити елемент на рядок
        strcat(s,"t"); // Долучити табуляцію після чергового елемента,
        fputs(s, f); // записати число з пробілом у файл
      }
      fputs("\n", f); // i символ [Enter]
  }
  fclose(f);
  cin.get();
  return 0;
}

```

Результати роботи програми (ліворуч – вигляд консольного вікна з результатами, праворуч – текстовий файл у Блокноті):

77	76	64	44	37	Файл	Правка	Вид	Кодировка	Справка	100 %
-7	33	38	-5	24	77	76	64	44	37	^
11	-10	32	78	5	-7	33	38	-5	24	
68	-8	42	-3	16	11	-10	32	78	5	
66	-1	-11	67	77	68	-8	42	-3	16	
					66	-1	-11	67	77	v
					<					>

Для виведення одновимірного масиву з текстового файлу слід зчитати рядок і розбити його на числа, наприклад, за допомогою функції `strtok()`:

```
char s[80];
char *D=" "; // Пробіл – розділювач між числами у рядку
char *p=new char[80]; // Рядок для числа у символьній формі
fgets(s,80,f); // Зчитування рядка,
p=strtok(s,D); // і виокремлення першого числа.
while (p) // Допоки рядок не є порожній,
{ a[i]=atoi(p); // перетворюється ціле число із символьної форми,
  i++; // збільшується індекс елемента масиву
  p=strtok(NULL,D); // і виокремлюється наступне число.
}
```

Приклад 12.7. Заповнити матрицю 3×5 числами з текстового файлу й обчислити суму елементів матриці.

Розв'язок. У Блокноті слід створити текстовий файл з ім'ям `matr.txt`, в якому записати елементи матриці:

Файл	Правка	Вид	Кодировка	Справка	100 %
2,5	0,4	5,5	8,8	-3,4	
2,1	1,7	-4,4	4,6	0,5	
4,5	6,3	0,8	-8,7	3,9	

Текст програми:

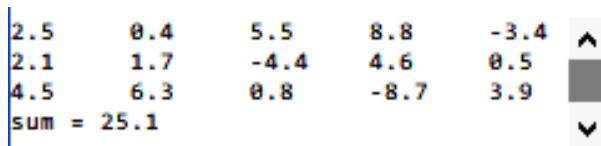
```
#include <iostream>
using namespace std;
int main()
{
    system("color F0");
    double m[3][5], sum=0;
    FILE *f;
    char *s=new char[50];
    int i=0, j;
    char *p=new char [50];
    if ((f=fopen("matr.txt", "rt"))==0)
    { cout<<"Файл не вдалось відкрити"; cin.get(); return 0; }
    while (fgets(s,50,f)) // У циклі зчитуються рядки з файлу.
    { j=0; // Розпочинаючи з нульового елемента,
      p=strtok(s, " "); // виокремлюється перше число
      while (p) // і, допоки у рядку є числа,
      { m[i][j]=atof(p); // перетворюється елемент із символьної форми,
        j++; // збільшується індекс стовпчика
        p=strtok(NULL, " "); // і виокремлюється наступне число
      }
      i++; // Збільшення індексу рядка (перехід до нового рядка матриці)
    }
    fclose(f);
```

```

for (int i1=0; i1<i; i1++)
{ for (int j1=0; j1<j; j1++)
  { cout << m[i1][j1] << "\t";
    sum+=m[i1][j1];
  } cout << endl;
}
cout<<"sum = "<<sum<<endl;
cin.get();
return 0;
}

```

Результати роботи програми:



```

2.5      0.4      5.5      8.8      -3.4
2.1      1.7     -4.4      4.6      0.5
4.5      6.3      0.8     -8.7      3.9
sum = 25.1

```

12.2.2 Робота з текстовими файловими потоками у стилі C++

Стандартна бібліотека C++ містить три класи потоків для роботи з файлами:

ifstream	вхідні файлові потоки (для зчитування)
ofstream	вихідні файлові потоки (для записування)
fstream	двонаправлені файлові потоки (для зчитування та записування)

При роботі з файлами цих класів треба долучати до програми заголовний файл `<fstream>`.

Об'єкти файлових потоків створюються за допомогою конструкторів відповідних класів, наприклад:

```

// Створення (відкриття) вихідного потоку (записування)
ofstream outfile("Test.dat");
// Створення (відкриття) вхідного потоку (зчитування)
ifstream fin ("Test.dat");
// Створення (відкриття) введення-виведення (записування і зчитування)
fstream f_in_out("Test.dat");

```

До створених у такий спосіб потоків можна застосовувати операції «помістити в потік» (<<) і «взяти з потоку» (>>). Перевага цих операцій, які працюють з текстовими файлами, порівняно з розглянутими у попередніх розділах функціями, є простота використання і автоматичне розпізнавання типів даних. Розглянемо, наприклад, такий код:

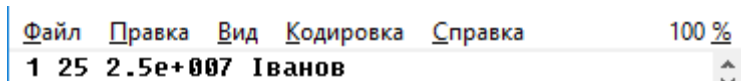
```

int i=1, j=25, i1, j1; double a=25e6, a1; char s[40], s1[40];
strcpy(s, "Іванов");
ofstream fout ("Test.txt"); // Створити файл як вихідний потік
if (!fout) {cout<<"Файл не вдалось створити"; cin.get(); return 0;}
fout << i << ' ' << j << ' ' << a << ' ' << s <<endl;
fout.close(); // Закрити файл
ifstream fin("Test.txt"); // Відкрити файл як вхідний потік

```

```
if (!fin) {cout<<"Файл не вдалось відкрити"; cin.get(); return 0;}
fin >> i1 >> j1 >> a1 >> s1;
cout<< i1 << '\t'<<j1 << '\t'<< a1<< '\t' << s1;
fin.close();                // Закрити файл
```

Вміст текстового файлу Test.txt після цих команд у редакторі Блокнот:



Наприкінці записаного рядка у цьому файлі маніпулятором потоку endl здійснюється перехід до нового рядка. Причому записування всіх різнотипних даних рядка здійснюється одним оператором, що містить зчеплені операції «помістити у потік». Якщо порівняти це з аналогічними кодами, наведеними у попередніх підрозділах, то можна переконатися в компактності й простоті застосування цієї операції. Наступні після закривання файлу команди створюють вхідний потік, пов'язаний з цим файлом, і за допомогою операції «узяти з потоку» читають дані.

Як було зазначено раніш, файл треба описати. У C++ це можна здійснити в такий спосіб:

```
ofstream filename("C:\Test.dat", ios::out);
```

Тут у подвійних лапках зазначено ім'я файлу на диску C, а filename – ім'я файлової змінної, тобто ім'я, за допомогою якого здійснюватиметься відкривання файлу у програмі. Якщо файл не існує, то його буде створено. Далі, після коми, слід зазначити режим відкривання файлу (табл. 12.3).

Таблиця 12.3

Режими відкривання файлів

Режим	Значення режиму
ios::in	Відкрити файл для зчитування
ios::out	Відкрити файл для записування
ios::trunc	Вилучити дані з файлу
ios::nocreate	Якщо файл не існує, відкривання файлу не відбудеться
ios::ate	Після відкриття перемістити позицію зчитування-записування на кінець файлу
ios::app	Відкрити для записування даних у кінець файлу
ios::binary	Відкрити файл, вважаючи його за бінарний (не текстовий)

Режими ios::ate та ios::app є схожими, позаяк переміщують позицію зчитування-записування (файловий вказівник) у кінець файлу. Відмінність між ними полягає в тому, що режим ios::app дозволяє дописувати дані лише у кінець файлу, в той час як режим ios::ate просто переміщує вказівник на кінець файлу за аналогією роботи C-функції fseek(f, 0, SEEK_END).

Імена констант відкриття є аббревіатурами виконуваних дій: app – append (долучити), ate – at end (на кінець), trunc – truncate (урізати, відкинути) тощо.

Режими відкриття файлу за суттю є бітовими масками, а тому можна задавати два та більше режимів, поєднуючи їх операцією логічного «АБО». На-

приклад, щоб відкрити бінарний файл з ім'ям `Test.dat` як для зчитування, так і для записування, слід записати функцію

```
fstream filename("Test.dat", ios::in|ios::out|ios::binary);
```

Тут символ `|` поєднує два режими `ios::in` та `ios::out`.

У табл. 12.4 наведено відповідність поміж режимами відкривання у C++ і C.

Таблиця 12.4

Відповідність режимів відкривання файлів в C++ і C

Режим C++	Режим C	Значення режиму
<code>ios::in</code>	<code>"r"</code>	Відкрити файл для зчитування. Позиція встановиться на початок файлу
<code>ios::out</code> <code>ios::out ios::trunc</code>	<code>"w"</code>	Відкрити файл для записування. Позиція встановиться на початок файлу. Якщо файл існує, записування здійснюватиметься поверх даних, тобто попередньо відбудеться урізання довжини файлу до нуля
<code>ios::out ios::app</code>	<code>"a"</code>	Відкрити файл для записування даних у кінець файлу
<code>ios::in ios::out</code>	<code>"r+"</code>	Відкрити файл для зчитування-записування даних, починаючи з довільної позиції, тобто позиція може змінюватись
<code>ios::in ios::out ios::trunc</code>	<code>"w+"</code>	Відкрити файл для зчитування-записування, урізавши довжину файлу до нуля

Щоб перевірити правильність відкриття файлу, можна скористатися умовним оператором:

```
if (!filename)
{ cout<<"Файл не вдалось відкрити"; cin.get(); return 0;}
```

Завершення роботи з файлом здійснюється за допомогою функції:

```
filename.close();
```

Для записування певної інформації у файл використовується операція «помістити у потік» (за аналогією з `cout<<`):

```
filename << блок1 << блок2 << ... << блокN;
```

Для зчитування з файлу певної інформації використовується операція «взяти з потоку» (за аналогією з `cin>>`)

```
filename >> блок1 >> блок2 >> ... >> блокN;
```

Однак при читанні з файлу за допомогою потоків зчитування здійснюється до пробілу чи символу нового рядка. Тому це зчитування придатне лише для чисел та окремих слів. Для зчитування цілого рядка використовується функція `getline()`, яка має прототип:

```
getline (char*, int, char='\n');
```

Наприклад, оператор

```
fin.getline(s, n);
```

читає з потоку не більше за `n-1` символів й записує їх до змінної `s`.

Для роботи з файловим потоком слід до програми долучити бібліотеку `<fstream>`.

Розглянемо приклад, в якому показано створення файлу, послідовне записування та зчитування даних.

Приклад 12.8. Сформувати у Блокноті текстовий файл та заповнити його даними про студентів: прізвище, найменування групи, рік народження. Створити програмний код для переглядання всього списку і відбору даних про студентів з групи КТ-19.

Файл	Правка	Вид	Кодировка	Справка	100 %
Антонюк	КТ-19	2000			
Борисов	ПЗ-28	1999			
Васюк	КТ-28	1998			
Данилюк	КТ-19	1999			
Гладун	КТ-18	2000			
Логінов	ПЗ-18	2001			
Масимча	ПЗ-18	1997			

Розв'язок. Відбирання даних про студентів з групи КТ-19 організуємо у вигляді окремої функції, аргументом якої буде ім'я файлу.

Текст програми:

```
#include <iostream>
#include <fstream>
using namespace std;
void KT(char* fname)
{ char PIB[50], gr[10]; int god;
  puts("\tСтуденти КТ-19");
  ifstream f(fname);
  while(f >> PIB >> gr >> god)
    if (strcmp(gr, "КТ-19")==0) cout<<PIB<<" "<<gr<<" "<<god<<endl;
  f.close();
}
int main()
{ system("color F0"); setlocale(0, ".1251");
  char *s=new char[80], *fname="dan.txt";
  puts("\tДані з файлу");
  ifstream f(fname); // Відкрити файл
  if (!f)
    { cout<<"Файл не вдалось відкрити";
      cin.get(); return 0; }
  while (f.getline(s,80)) puts(s);
  f.close();
  KT(fname);
  cin.get();
  return 0;
}
```

Дані з файла		
Антонюк	КТ-19	2000
Борисов	ПЗ-28	1999
Васюк	КТ-28	1998
Данилюк	КТ-19	1999
Гладун	КТ-18	2000
Логінов	ПЗ-18	2001
Масимча	ПЗ-18	1997
Студенти КТ-19		
Антонюк	КТ-19	2000
Данилюк	КТ-19	1999

До речі, записувати дані у файл і зчитувати їх можна з довільної позиції файлу. Позиціонування (встановлення курсора на певну позицію) виконується за допомогою методу `seekp()` класу `ifstream` при записуванні даних у файл, і за допомогою методу `seekg()` класу `ofstream` – при зчитуванні даних з файлу. Ці методи мають два аргументи: перший зазначає на скільки байтів від позиції, зазначеної у другому аргументі, слід зрушитися. Існують такі константні значення позицій:

`ios::beg` – початок потоку (встановлюється за промовчанням);

`ios::end` – кінець потоку;

`ios::cur` – поточна позиція потоку.

Наприклад:

`f.seekg(n);` – позиціонування на *n*-й байт від початку файлу;

`f.seekg(n, ios::cur);` – позиціонування на *n*-й байт уперед від поточної позиції;

`f.seekp(k, ios::end);` – позиціонування на *k*-тий байт від кінця файлу;

`f.seekp(0, ios::end);` – позиціонування на кінець файлу.

Методи `tellg()` та `tellp()` повертають поточну позицію файлу для потоків введення та виведення відповідно.

Для створення текстових файлів довільного доступу існує багато способів. Мова C++ не накладає вимог щодо вмісту текстових файлів, але записані у файлі рядки мають бути однакової фіксованої довжини. Це дає змогу легко визначати точне місцезнаходження будь-якого рядка щодо початку файлу. У такому файлі дані може бути записано в будь-яке місце та змінено без перезаписування всього файлу. Для організації такого записування використовується маніпулятор `setw(num)`, який задає довжину поля виведення з *num* позицій. Для його використання слід долучити заголовний файл `<iomanip.h>`. Приклад його використання:

```
fout<<setw(15)<<name<<setw(20)<<surname<<setw(7)<<year<<endl;
```

12.2.3 Опрацювання текстових файлів засобами простору імен System::IO платформи .NET Framework

Деякі класи простору System::IO для роботи з файлами

Простір імен `System::IO` платформи `.NET Framework` містить декілька класів, які дозволяють здійснювати операції з потоками даних, файлами і папками (каталогами), такі як читання і записування даних, пошук (визначення і змінення поточної позиції всередині потоку), архівація файлів і каталогів тощо. Наприклад, клас `StreamWriter` з цього простору імен має сучасні, потужні і доволі зручні засоби записування даних у файли як потоки даних.

Охарактеризуємо деякі з цих класів.

Клас	Опис
File	надає методи для створення, копіювання, видалення, переміщення і відкриття файлів
FileInfo	надає методи екземпляра для створення, копіювання, видалення, переміщення і відкриття файлів
Directory	надає статистичні методи для створення і переміщення у каталогах
DirectoryInfo	надає методи екземпляра для створення і переміщення у каталогах
Path	виконує операції щодо відомостей про шлях до файлу чи каталогу (літеральний рядок (тип String), який вказує розташування файлу у файловій системі – адресу каталогу)
StreamReader	надає засоби зчитування послідовності символів (байтів) з потоку в певному кодуванні
StreamWriter	надає засоби записування послідовності символів (байтів) у потік в певному кодуванні
BinaryReader	надає засоби зчитування простих типів даних (двійкових значень)
BinaryWriter	надає засоби записування простих типів даних (двійкових значень)

Розглянемо найпоширеніші методи цих класів, які дозволять створити текстовий файл, записати у нього дані, прочитати їх та ін.

Поширені дії при роботі з файлами

Дії	Методи
Створити текстовий файл	File::CreateText або File::Create FileInfo::CreateText або FileInfo::Create
Відкрити файл у заданому режимі і доступі	File::Open File::OpenRead – відкрити файл у режимі читання; File::OpenWrite – відкрити файл для записування
Записати у текстовий файл	Write – записати у потік текстове подання аргументів; WriteLine – записати рядок і символ '\n' як ознаку кінця рядка; File::WriteAllText – відкрити текстовий файл, записати всі дані як один рядок у файл і закрити файл
Дописати дані у текстовий файл	File::AppendText FileInfo::AppendText методи відкривають файл і дописують в нього текст у кодуванні UTF-8 (якщо файлу не існувало, він буде створений)
Зчитати з текстового файлу	Read – зчитати з потоку текстове подання аргументів, записаних у дужках; ReadLine – зчитати рядок і символ '\n' як ознаку кінця рядка; File::ReadAllText – відкрити текстовий файл, зчитати всі рядки файлу в один зазначений рядок і закрити файл
Закрити файл (потік)	Close

Дії	Методи
Перемістити позицію файлу на offset байтів від позиції SeekOrigin	Stream::Seek(offset, SeekOrigin)
Повернути наступний доступний символ, але не використовувати його	StreamReader::Peek з цим методом можна організувати цикл для почергового опрацювання вмісту файлу: while (fr->Peek() > -1) . . .
Перейменувати чи перемістити файл	File::Move або FileInfo::MoveTo
Видалити файл	File::Delete або FileInfo::Delete
Копіювати файл	File::Copy або FileInfo::CopyTo
Визначити наявність файлу	File::Exists
Отримати дані про розмір файлу	властивість FileInfo::Length
Отримати розширення імені файлу	Path::GetExtension
Отримати повний шлях до файлу	Path::GetFullPath
Отримати ім'я й розширення файлу	Path::GetFileName
Змінити розширення файлу	Path::ChangeExtension

Тобто для однієї дії при роботі з файлами існують декілька методів її реалізації. Розглянемо на прикладах деякі з цих методів.

Створити текстовий файл з ім'ям `MyTest.rtf` у поточному каталозі можна командою:

```
StreamWriter^ f = File::CreateText("MyTest.rtf");
```

або командою:

```
TextWriter^ f = gcnew StreamWriter(File::OpenWrite("MyTest.rtf"));
```

Якщо створюваний файл слід розмістити не в поточному каталозі, а в якомусь іншому, то слід зазначити повний шлях до файлу:

```
StreamWriter^ f = File::CreateText("c:\\temp\\MyTest.rtf");
```

Записати у файл рядок s можна за допомогою методу `WriteLine`:

```
String^ s = "Текст, який буде записано у файл окремим абзацом.";
f->WriteLine(s);
```

Записати відразу цілий набір рядків, наприклад, вміст елемента `richTextBox1`, у файл з ім'ям `MyTest.rtf` у поточному каталозі, можна командою:

```
File::WriteAllText("MyTest.rtf", richTextBox1->Text);
```

Дописати дані у кінець файлу можна командою

```
File::AppendAllText("MyTest.rtf", "Текст, який буде дописано у файл\n");
```

При цьому, якщо файл з таким ім'ям не існував, він буде створений, тобто метод `AppendAllText` і створює файл, і відкриває його, і дописує в нього дані, поєднуючи можливості відразу декількох методів.

Для **зчитування рядків** текстового файлу слід спочатку створити екземпляр класу `StreamReader`, після чого по черзі зчитувати всі рядки з файлу:

```
StreamReader^ f = File::OpenText("MyTest.rtf");
String^ s = "";
while (f->Peek() > 0)
{ s = f->ReadLine(); Console::WriteLine( s ); }
```

У наведеному циклі умовою припинення зчитування рядків з файлу є ну-

льове значення довжини зчитаного рядка. Така конструкція операторів дозволить переглянути вміст всього текстового файлу у консольному режимі.

Іншим, більш компактним способом переглядання вмісту відразу всього текстового файлу є використання методу `File::ReadAllText`, який можна використовувати як у консолі, так і для елементів на формі. До того ж, для цього методу не треба попередньо використовувати команду відкриття файлу. Так для консолі програмний код для переглядання усього файлу може складатися лише з однієї команди:

```
Console::WriteLine(File::ReadAllText(fname));
```

А для проекту з формою, на якій передбачено елемент `richTextBox1` для виведення вмісту файлу команда може бути такою:

```
richTextBox1->Text = File::ReadAllText(fname);
```

Переміщуватися по файлу можна не лише при зчитуванні даних, а й за допомогою методу `Seek`, наприклад:

```
f->BaseStream->Seek(0, SeekOrigin::Begin); // перейти на початок файлу
```

```
f->BaseStream->Seek(0, SeekOrigin::End); // перейти у кінець файлу
```

Закрити файл доречно командою

```
f->Close();
```

Приклад 12.9. Створити файл з розширенням `txt`, в який ввести рядки з будь-якими словами (бажано не абракадабру) (кінець введення – порожній рядок). Запрограмувати переглядання створеного файлу, обчислення кількості рядків, в яких перше слово має парну кількість символів (довжину). Крім того, програмно вивести рядки, які містять великі літери.

Текст програми:

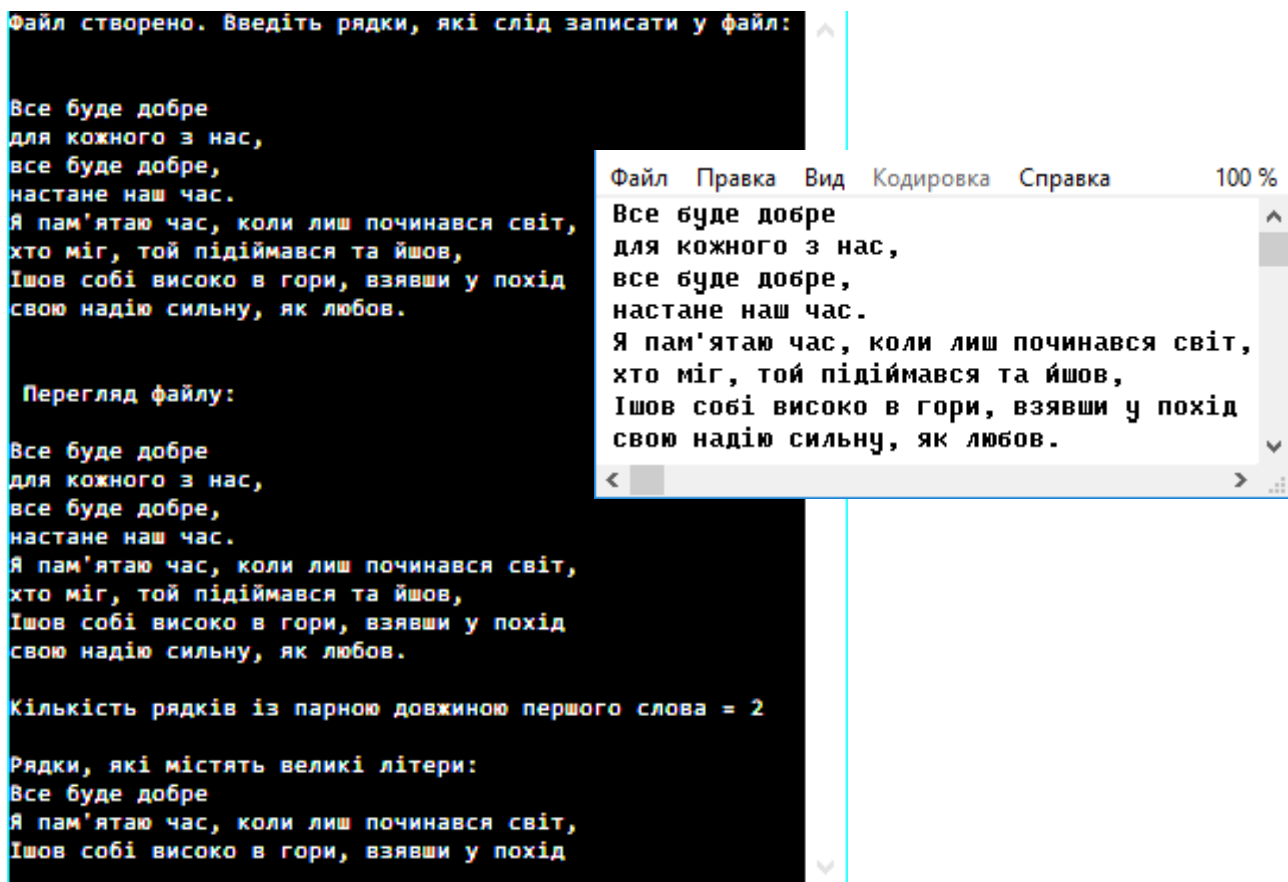
```
using namespace System;
using namespace System::IO;
int main(array<System::String ^> ^args)
{ String ^s, ^fname = "Якобчук.txt";
  StreamWriter^ fw = File::CreateText( fname ); // створити файл
  Console::WriteLine("Файл створено. Введіть рядки, які слід записати у файл:\n");
  do { s=Console::ReadLine(); // зчитати введений рядок
      fw->WriteLine(s); // записати рядок у файл
  } while (s!="");
  fw->Close(); // закрити файл
  StreamReader^ fr = File::OpenText( fname ); // відкрити файл для читання
  Console::WriteLine("\n Перегляд файлу:\n");
  while ( s = fr->ReadLine() ) // читати по черзі рядки з файлу
      Console::WriteLine( s ); // і виводити їх на екран
  fr->BaseStream->Seek(0, SeekOrigin::Begin); // перейти на початок файлу
  int kol=0, ind, n;
  while ( s = fr->ReadLine() )
  { ind = s->IndexOf(" "); // шукати індекс пробілу
    if (ind == -1 && s->Length>0) n=s->Length; // якщо рядок з одного слова
    else n=ind; // n – довжина першого слова
```

```

    if (n % 2 == 0) kol++;
}
Console.WriteLine("Кількість рядків із парною довжиною першого слова = "+kol);
fr->BaseStream->Seek(0, SeekOrigin::Begin);    // перейти на початок файлу
Console.WriteLine("\nРядки, які містять великі літери:");
while ( s = fr->ReadLine() )
{ for (int i=0; i<s->Length; i++)
    if (Char::IsUpper(s[i])) { Console.WriteLine(s); break; }
}
fr->Close();    Console::Read();
return 0;
}

```

Результат виконання програми:



Приклад 12.10. Створити файл з розширенням `rtf`, в який ввести рядки з будь-якими словами (бажано не абракадабру) (кінець введення – порожній рядок). Запрограмувати можливість переглядання і дописування даних до і наявного файлу. Програмно вивести слова текстового файлу, які містять більше однієї літери 'а', та обчислити їхню кількість. Вивести рядок, який містить найдовше слово.

Розв'язок. На відміну від попереднього прикладу програми, цю програму організуємо більш структуровано, організувавши різні етапи розв'язання в окремих функціях. Крім того, ця програма дозволить не створювати текстовий файл щоразу заново, а лише за потреби дописувати у нього дані.

Текст програми:

```
#include <iostream>
using namespace System;
using namespace System::IO;

void Записати(String^ fname)
{ String ^s;
  StreamWriter ^fw = File::AppendText( fname ); // відкрити чи створити файл
  try
  { do
    { s = Console::ReadLine(); // зчитати введений рядок
      if(s != L"") fw->WriteLine(s); // записати рядок у файл
    } while (s != "");
  }
  finally
  { fw->Close(); } // закрити файл
  return;
}

void Перегляд(String ^fname)
{ StreamReader ^fr = File::OpenText( fname ); // відкрити файл
  try
  { String ^s = "";
    // читати по черзі усі рядки з файлу і виводити їх на екран
    while ( s = fr->ReadLine() ) Console::WriteLine( s );
  }
  finally
  { fr->Close(); }
  return;
}

void Завдання1(String ^fname)
{ StreamReader ^fr = File::OpenText( fname );
  try
  { String ^s, ^delimStr = " ,.:\\t";
    array<Char> ^delimiter = delimStr->ToCharArray( );
    array<String^> ^words;
    int kol=0, ka;
    Console::WriteLine("\\nСлова, які містять більше однієї літери 'a': ");
    while ( s = fr->ReadLine() )
    { s = s->Trim();
      words = s->Split(delimiter, StringSplitOptions::RemoveEmptyEntries);
      for (int i=0; i<words->Length; i++)
      { for (int j=ka=0; j<words[i]->Length; j++)
        { if (words[i][j] == L'a' || words[i][j] == L'A' ||
            words[i][j] == L'a' || words[i][j] == L'A') ka++;
          if(ka > 1) { kol++; Console::WriteLine(words[i]);}
        }
      }
      Console::WriteLine("Кількість слів, які містять більше однієї літери 'a' = "+kol);
    }
  }
}
```



```

    finally
    { fr->Close(); }
    return;
}

void Завдання2(String^ fname)
{
    StreamReader ^fr = File::OpenText( fname );    // відкрити файл
    try
    { int max=0, n, k=0;
      String ^s, ^slovo, ^delimStr = " ,.:\\t";
      array<Char> ^delimiter = delimStr->ToCharArray( );
      array<String^> ^words;
      while ( s = fr->ReadLine() )
      { k++;                                // номер поточного рядка у файлі
        s = s->Trim();                      // видалити початкові і кінцеві пробіли
        words = s->Split(delimiter, StringSplitOptions::RemoveEmptyEntries);
        for (int i=0; i<words->Length; i++)
        { if (words[i]->Length > max)
          { max = words[i]->Length;        // запам'ятати довжину найдовшого слова
            slovo = words[i];              // запам'ятати найдовше слово
            n = k;                          // і номер рядка з найдовшим словом
          }
        }
      }
      fr->BaseStream->Seek(0, SeekOrigin::Begin); // перейти на початок файлу
      k=0;                                     // номер поточного рядка
      Console::WriteLine("\\nРядок з найдовшим словом - "+slovo+
        " (довжина - "+slovo->Length+"):");
      while ( s = fr->ReadLine() )
      { k++;
        if (k == n) { Console::WriteLine( s ); break; }
      } }
    finally
    { fr->Close();
    }
    return;
}

int main(array<System::String^> ^args)
{ system("color F0");
  String ^fname = "Сидоренко.rtf";
  if ( !File::Exists( fname ))
    Console::WriteLine("Файл створено. Введіть рядки, які слід записати у нього:\\n");
  else Console::WriteLine("Файл існує. Введіть рядки, які слід дописати у нього.
    \\nЯкщо не треба дописувати нові рядки, двічі натисніть Enter.\\n");
  Записати(fname);
  Console::WriteLine("Перегляд файлу:\\n");
  Перегляд(fname);
  Завдання1(fname);
}

```

```
Завдання2(fname);  
Console::Read();  
return 0;  
}
```

Результати виконання програми:

Файл створено. Введіть рядки, які слід записати у нього:

Рече та стогне Дніпр широкий,
Сердитий вітер завива,
Додолю верби гне високі,
Горами хвилю підійма.

Перегляд файлу:

Рече та стогне Дніпр широкий,
Сердитий вітер завива,
Додолю верби гне високі,
Горами хвилю підійма.

Слова, які містять більше однієї літери 'а':
завива
Кількість слів, які містять більше однієї літери 'а' = 1

Рядок з найдовшим словом - Сердитий (довжина - 8):
Сердитий вітер завива,

Файл існує. Введіть рядки, які слід дописати у нього.
Якщо не треба дописувати нові рядки, двічі натисніть Enter.

Ще треті півні не співали,
Ніхто нігде не гомонів,
Сичі в гаю перекликались,
Та ясен раз у раз скрипів.

Перегляд файлу:

Рече та стогне Дніпр широкий,
Сердитий вітер завива,
Додолю верби гне високі,
Горами хвилю підійма.
Ще треті півні не співали,
Ніхто нігде не гомонів,
Сичі в гаю перекликались,
Та ясен раз у раз скрипів.

Слова, які містять більше однієї літери 'а':
завива
Кількість слів, які містять більше однієї літери 'а' = 1

Рядок з найдовшим словом - перекликались (довжина - 13):
Сичі в гаю перекликались,

The screenshot shows a Windows environment with a console window and a Microsoft Word window. The console window displays the output of a program that creates a file, writes text to it, and then analyzes the file for specific characters and word lengths. The Word window shows the content of the file as it appears in a text editor.

12.2.4 Зчитування і записування текстових файлів методами елементів керування Visual C++

Робота з текстовими файлами може здійснюватися за допомогою методів ReadAllText() та WriteAllText() простору імен System.IO, які є у класу File. Ці класи описують списки рядків і мають безліч методів маніпулювання рядками.

Для того щоб прочитати (вивантажити) вміст певного текстового файлу з ім'ям "example.txt", створеного, наприклад у Блокноті, у вікно richTextBox1 можна скористатися командою:

```
richTextBox1->Text = File::ReadAllText("example.txt");
```

Вивантажити вміст вікна richTextBox1 у файл можна командою:

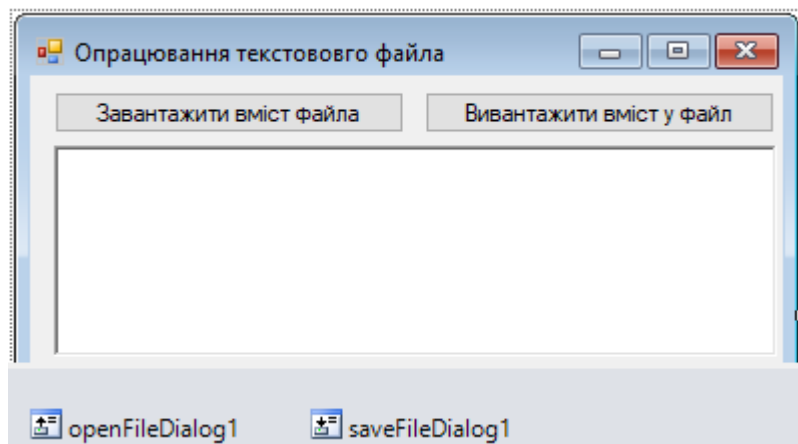
```
File::WriteAllText("example.txt", richTextBox1->Text,  
                  System::Text::Encoding::UTF8);
```

При відкриванні та зберіганні файлу можна користуватися стандартними елементами-діалогами openFileDialog та saveFileDialog. Для відкривання, переглядання й редагування файлу можна користуватися багаторядковим елементом richTextBox. При цьому можна працювати з файлами у форматах TXT та RTF. Властивості Lines та Text елемента мають тип String[] Array, що дозволяє безпосередньо користуватися методами LoadFile(), SaveFile(), ReadAllText() та WriteAllText(), наприклад:

```
richTextBox1->LoadFile(fileName);  
richTextBox1->SaveFile(fileName);
```

Через елементи керування Visual C++ можна працювати не лише з текстовими файлами, але і з файлами зображень і мультимедіа.

Наведемо найпростіший приклад завантаження текстового файлу і зберігання змінень, якщо їх було внесено. Для цього на формі розмістимо елементи richTextBox1, openFileDialog1, saveFileDialog1 та дві кнопки button:



Текст програми з формою у Visual C++ :

```
// Кнопка «Завантажити вміст файла»  
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)  
{  
    if (openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)  
        richTextBox1->LoadFile(openFileDialog1->FileName);  
}
```

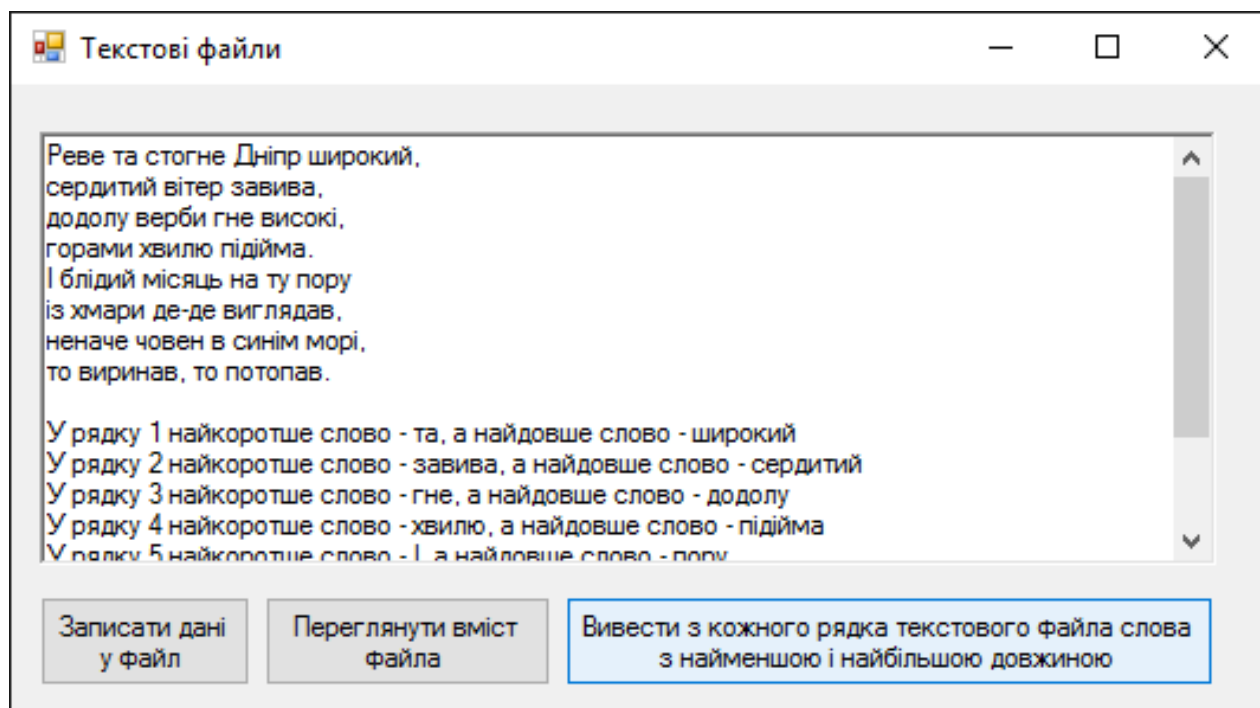
```
// Кнопка «Вивантажити вміст у файл»
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e)
{
    if (saveFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)
        richTextBox1->SaveFile(saveFileDialog1->FileName);
}
```

Приклад 12.11. Створити текстовий файл, в який ввести рядки з будь-якими словами (бажано не абракадабру). Передбачити можливість переглядання створеного файлу та вивести з кожного рядка найкоротше та найдовше слова.

Текст програми:

```
using namespace System::IO;
// Записати дані у файл
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{
    File::WriteAllText("example.txt", richTextBox1->Text,
                      System::Text::Encoding::UTF8);
}
// Переглянути вміст файлу
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e)
{
    richTextBox1->Text = File::ReadAllText("example.txt");
}
// Вивести з кожного рядка файлу слова з найменшою і найбільшою довжиною
private: System::Void button3_Click(System::Object^ sender, System::EventArgs^ e)
{
    StreamReader ^f = File::OpenText( "example.txt" );
    array<Char> ^separator = { ' ', '.', ',', '!', '?', '\t' };
    try
    {
        String ^s, ^smin, ^smax; int kmin, kmax, k=0;
        while (s = f->ReadLine())
            if (s->Length > 0)
            { k++;
              array<String^> ^subs = s->Split(separator,
                                              StringSplitOptions::RemoveEmptyEntries);
              kmin=kmax=subs[0]->Length; smin=smax=subs[0];
              for(int i=0; i<subs->Length; i++)
              { if (kmin > subs[i]->Length) smin=subs[i];
                if (kmax < subs[i]->Length) smax=subs[i];
              }
              richTextBox1->Text+="У рядку "+k.ToString()+" найкоротше слово - " +
                                smin+", а найдовше слово - "+smax+"\n";
            }
        finally
        { f->Close(); }
    }
}
```

Результат виконання програми:



12.3 Бінарні файли

У **бінарному** (двійковому) файлі число, на відміну від текстового, зберігається у внутрішньому його поданні. У двійковому форматі можна зберігати не лише числа, а й рядки та цілі інформаційні структури. Причому останні зберігати зручніше, завдяки тому що відсутня потреба явно зазначати кожен елемент структури, а зберігається вся структура як цілковита одиниця даних. Хоча цю інформацію не можна прочитати як текст, вона зберігається більш компактно і точно. Тому, що саме і в якій послідовності розміщено в бінарному файлі, має бути відомо програмі.

12.3.1 Робота з бінарними файлами у стилі C

З двійковими файлами можна виконувати ті ж самі дії, що і з текстовими. Для відкривання бінарного файлу використовується та сама команда `fopen()`, лише у другому параметрі (режимі відкривання файлу) замість літери «t» треба записати літеру «b». Наприклад, бінарний файл з ім'ям `tmp.dat` можна відкрити для зчитування такою командою:

```
f = fopen("tmp.dat", "rb");
```

де `f` – вказівник типу `FILE*`.

Записування і зчитування у двійкових файлах найчастіше здійснюються за допомогою відповідно функцій `fwrite()` та `fread()`.

Прототип функції зчитування `fread()`:

```
size_t fread
( char *buffer,           // Масив для зчитування даних,
  size_t elemSize,        // розмір одного елемента,
```

```
    size_t numElems,    // кількість елементів для зчитування
    FILE *f             // і вказівник потоку
);
```

Тут `size_t` означений як беззнаковий цілий тип у системних заголовних файлах. Функція намагається прочитати `numElems` елементів з файлу, який задається вказівником `f`, розмір кожного елемента становить `elemSize`. Функція повертає реальну кількість прочитаних елементів, яка може бути менше за `numElems`, у разі завершення файлу чи то помилки зчитування.

Приклад використання функції `fread()`:

```
FILE *f;
double buff[100];
size_t res;
f = fopen ("tmp.dat", "rb");    // Відкриття файлу
if (f == 0)
{   cout<<"Не вдається відкрити файл для зчитування"<<endl;
    exit (1);                  // Завершення роботи з кодом 1
}
// Зчитування 100 дійсних чисел з файлу
res = fread(buff, sizeof(double), 100, f);
// res дорівнює реальній кількості прочитаних чисел
```

У даному прикладі файл `tmp.dat` відкривається для зчитування як бінарний, з нього читаються 100 дійсних чисел розміром 8 байтів кожне. Функція `fread()` повертає реальну кількість прочитаних чисел, яка є менше чи то дорівнює 100.

Функція `fread()` читає інформацію у вигляді потоку байтів і в незмінному вигляді розміщує її в пам'яті. Слід розрізнявати текстове подавання чисел і їхнє бінарне подавання. У наведеному вище прикладі числа у файлі має бути записано у *бінарному форматі*, а не у вигляді тексту. Для текстового введення чисел слід використовувати функції введення за форматом, які було розглянуто вище.

Функція бінарного записування у файл `fwrite()` є аналогічна до функції зчитування `fread()`. Вона має такий прототип:

```
size_t fwrite
( char *buffer,        // Масив записуваних даних,
  size_t elemSize,     // розмір одного елемента,
  size_t numElems,     // кількість записуваних елементів
  FILE *f              // і вказівник потоку
);
```

Функція повертає кількість реально записаних елементів, яка може бути менше за `numElems`, якщо при записуванні виникла помилка: наприклад, не вистачило вільного простору на диску.

Приклад використання функції `fwrite()`:

```
FILE *f;
double buff[100];
size_t res;
. . . // Введення елементів масиву buff[100]
f = fopen("tmp.dat", "wb"); // Створення бінарного файлу
```



```

if (f == 0)
{ cout<<"Не вдається відкрити файл для запису"<<endl;
  cin.get();
  exit(1);      // Завершення роботи програми з кодом 1
}
// Записування 100 дійсних чисел у файл
res = fwrite(buff, sizeof(double), 100, f);
// У разі успішного записування res = 100

```

До обох функцій передається вказівник на дані, які виводяться чи вводяться, параметр `elemSize` задає розмір передаваних даних у байтах, а параметр `numElems` визначає кількість таких даних.

Так само, як і в текстових, у бінарних файлах можна визначати позицію зчитування-записування і переміщувати її у довільне місце файлу командами відповідно `ftell()` та `fseek()`. Можливість переміщувати позицію є корисна для файлів, які складаються з однорідних записів однакового розміру. Наприклад, якщо у файлі записано лише дійсні числа типу `double`, то для того, щоб прочитати *i*-те число, слід виконати оператори:

```

fseek(F, sizeof(double)*(i-1), 0);
fread(&a, sizeof(double), 1, F);

```

У такий спосіб можна читати які завгодно записи у будь-якій послідовності. За допомогою переміщення позиції можна редагувати записи у файлі. Нехай, приміром, треба змінити у файлі одне з чисел, помноживши його на 10. Це можна зробити, якщо відкрити файл у режимі зчитування і записування (наприклад, у форматі `"rb+"`), переміститись у відповідну позицію і виконати оператори:

```

fread(&z, sizeof(double), 1, F);
z *= 10;
fseek(F, -sizeof(double), 1);
fwrite(&z, sizeof(double), 1, F);

```

Перший з цих операторів читає число з файлу у дійсну змінну *z* типу `double`, другий – помножує її на 10. Третій оператор повертає позицію на один запис назад, оскільки після виконання `fread()` позиція просунулась уперед. Останній оператор пише в ту позицію, з якої було прочитано число, нове значення.

Те ж саме завдання можна розв'язати в інший спосіб:

```

long int pos = ftell(F);      // Запам'ятовування позиції
fread(&z, sizeof(double), 1, F); // Зчитування числа з файлу до змінної
z *= 10;
fseek(F, pos, 0);             // Відновлення позиції
fwrite(&z, sizeof(double), 1, F); // Записування числа зі змінної z у файл

```

Тут функція `ftell()` визначає поточну позицію файлу, з якої читається число, а функція `fseek()` відновлює цю позицію перед записуванням зміненого числа.

Як було зазначено вище, за допомогою двійкових файлів можна записувати й зчитувати не лише числа та рядки, а й структури (див. далі приклад 12.13).

Приклад 12.12. Створити бінарний файл, який містить цілі числа, і віднайти серед них найбільше число.

Текст програми:

```
#include <iostream>
#include <stdio.h>
using namespace std;
void writing(char* fname) // Функція записування чисел у файл
{ FILE *f; // Оголошення файлової змінної
  int x;
  f=fopen(fname,"rb+"); // відкривання файлу для зчитування і записування
  if (f==NULL)
    f=fopen(fname,"wb+"); // якщо файл не вдалось відкрити, створити його
  fseek(f, 0, SEEK_END); // перейти у кінець файлу
  char yn; // ознака бажання дописувати нові рядки у файл
  setlocale(0, ".1251");
rep:
  cout<<"Будете вводити нове ціле число для записування у файл? у/н" <<endl;
  cin>>yn;
  if (yn!='y') { fclose(f); return; }
  cout<<"Введіть ціле число для записування у файл та натисніть Enter" <<endl;
  cin>>x;
  fwrite(&x, sizeof(int), 1, f); // Дописування у файл
  goto rep;
}
int look(char* fname) // Функція зчитування чисел з файлу
{ int x; FILE *f;
  if ((f=fopen("num.dat","rb"))==0) // Відкриття файлу для зчитування
  { cout<<"Не вдається відкрити файл для читання"<<endl;
    cin.get(); return 0;
  }
  fseek(f, 0, 0);
  setlocale(0, ".1251"); cout<<"\nВміст файлу:\n";
  while (fread(&x, sizeof(int),1,f)) // Зчитування числа
    cout<<x<<"\t";
  // Визначення розміру файлу
  fseek(f, 0, SEEK_END);
  int n=ftell(f);
  fclose(f);
  return n;
}
int maxEl(char* fname) // Функція пошуку максимального числа з файлу
{ int max,x; FILE *f;
  if ((f=fopen("num.dat","rb"))==0) // Відкриття файлу для зчитування
  { cout<<"Не вдається відкрити файл для читання"<<endl;
    cin.get(); return 0;
  }
}
```

```

    fseek(f, 0, 0);
    fread(&max, sizeof(int),1,f);
    while (fread(&x, sizeof(int),1,f)) // Зчитування числа
        if (max<x) max=x;
    fclose(f);
    return max;
}
int main()
{ system("color F0");
  char s[]="num.dat"; // Визначення імені файлу
  writing(s);
  int len=look(s);
  setlocale(0, ".1251");
  cout<<"\nРозмір файлу - "<< len << endl;
  cout<<"Max = "<<maxEl(s)<<endl;
  cin.get();
  return 0;
}

```

Результат виконання програми:

Будете вводити нове ціле число для записування у файл? y/n

y

Введіть ціле число для записування у файл та натисніть Enter

77

Будете вводити нове ціле число для записування у файл? y/n

n

Вміст файлу:

38 -23 90 -7 39 00 -45 9 3 -1 -4 18 77

Розмір файлу – 52

Max = 90

Приклад 12.13. Ввести відомості про екзаменаційні оцінки студентів: прізвище та оцінка. Вивести відомості про студентів, які отримали «незадовільно» (менше за 60 балів).

Текст програми:

```

#include <iostream>
#include <stdio.h>
using namespace std;
struct ved // Оголошення структури
{ char priz[15];
  int oc;
} v;
void writing(char* fname)
{ FILE *f; // Оголошення файлової змінної
  f=fopen(fname,"rb+"); // Відкривання файлу для зчитування і записування
  if (f==NULL) // Якщо файл не вдалось відкрити,
    f=fopen(fname,"wb+"); // створити його
}

```

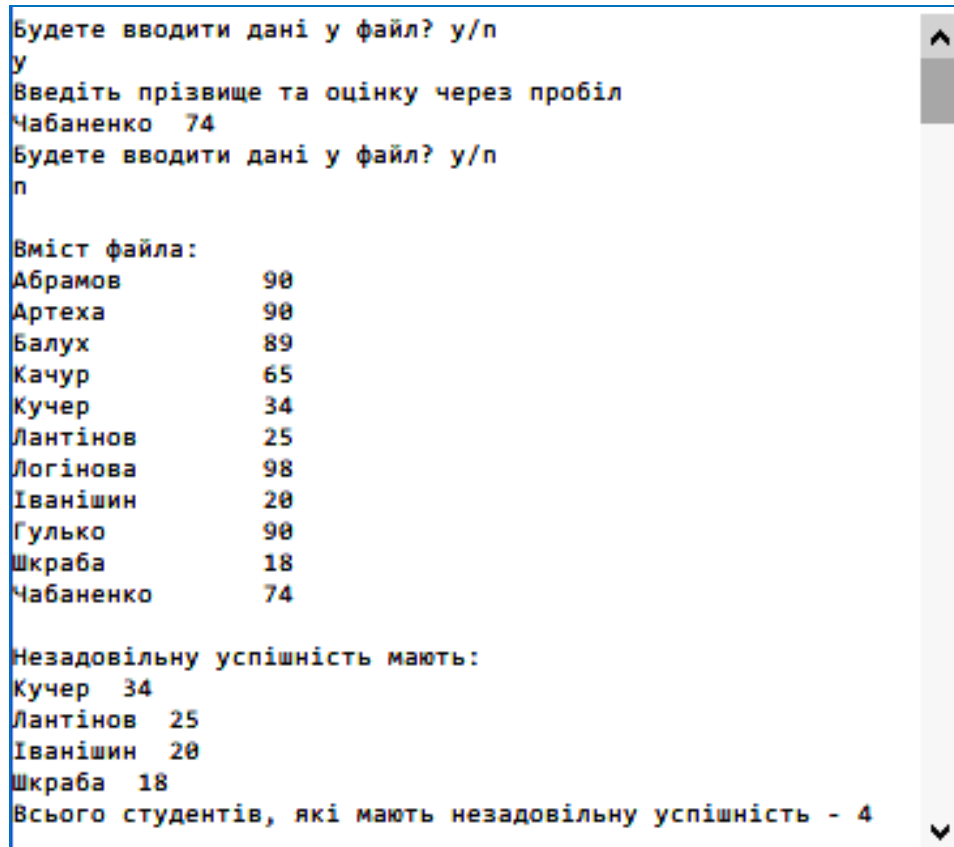
```
fseek(f, 0, SEEK_END); // Передід у кінець файлу
char yn;                // Ознака бажання дописувати нові рядки у файл
rep:
    setlocale(0, ".1251");
    cout<<"Будете вводити дані у файл? у/н"<<endl;
    cin>>yn;
    if (yn!='y') { fclose(f); return; }
    cout<<"Введіть прізвище та оцінку через пробіл"<<endl;
    setlocale(0, ".ОСР");
    cin>>v.priz>>v.oc;
    fwrite(&v, sizeof(v), 1, f); // Дописування структури у файл
    goto rep;
}

void look(char* fname)
{
    FILE *f;
    if ((f=fopen(fname, "rb"))==0) // Відкриття файлу для зчитування
    { cout<<"Не вдається відкрити файл для читання"<<endl;
      cin.get(); return;
    }
    setlocale(0, ".1251"); cout<<"\nВміст файлу:\n";
    setlocale(0, ".ОСР");
    while (fread(&v, sizeof(v), 1, f)) // Зчитування з файлу в змінну v
    { for (int i=strlen(v.priz); i<15; i++) strcat(v.priz, " ");
      cout<<v.priz<<" "<<v.oc<<endl;
    }
    fclose(f);
    return ;
}

void notpassed (char* fname)
{
    int k=0; FILE *f;
    if ((f=fopen(fname, "rb"))==0)
    { cout<<"Не вдається відкрити файл для читання"<<endl;
      cin.get();
      return;
    }
    setlocale(0, ".1251");
    cout<<"\nНезадовільну успішність мають:\n";
    setlocale(0, ".ОСР");
    while (fread(&v, sizeof(v), 1, f))
    { if (v.oc < 60)
      { cout << v.priz << " " << v.oc << endl;
        k++;
      }
    }
    setlocale(0, ".1251");
}
```

```
cout<<"Всього студентів, які мають незадовільну успішність - "  
    << k << endl;  
fclose(f);  
return;  
}  
int main()  
{  
    char s[]="ocinki.dat"; // і визначення імені файлу  
    writing(s);  
    look(s);  
    notpassed(s);  
    cin.get(); cin.get();  
    return 0;  
}
```

Результат виконання програми:



The screenshot shows a terminal window with the following text:

```
Будете вводити дані у файл? у/п  
у  
Введіть прізвище та оцінку через пробіл  
Чабаненко 74  
Будете вводити дані у файл? у/п  
п  
  
Вміст файла:  
Абрамов          90  
Артеха           90  
Балух            89  
Качур            65  
Кучер            34  
Лантінов         25  
Логінова         98  
Іванішин         20  
Гулько           90  
Шкраба           18  
Чабаненко        74  
  
Незадовільну успішність мають:  
Кучер 34  
Лантінов 25  
Іванішин 20  
Шкраба 18  
Всього студентів, які мають незадовільну успішність - 4
```

Приклад 12.14. Відсортувати дані файлу з відомостями про екзаменаційні оцінки (який було створено у попередньому прикладі) за спаданням оцінок.

Розв'язок. Вважаємо, що файл `ocinki.dat` вже створено і розміщено у папці з проектом. Оскільки при сортуванні файлу треба кілька разів переходити до початку файлу, а всі операції з файлом виконуються набагато повільніше, ніж з масивом, є сенс записати всі дані з файлу до динамічного масиву, відсортувати його, а потім повернути дані з масиву до файлу.

Текст програми (код функції **look** наведено у прикладі 12.13):

```
#include <iostream>
#include <stdio.h>
using namespace std;
struct ved // Оголошення структури
{ char priz[15];
  int oc;
} v;
void sort(char*fname)
{
    FILE *f;
    int i=0, k=0; // k - кількість записів у файлі
    bool pr; // Ознака - чи відсортовано масив
    setlocale(0, ".1251");
    if ((f=fopen(fname, "rb"))==0) // Відкриття файлу для зчитування
    { cout<<"Не можу відкрити файл для читання"<<endl; cin.get(); return; }
    setlocale(0, ".OCP");
    while (fread(&v, sizeof(ved), 1, f))
        k++; // Обчислення кількості записів у файлі
    fseek(f, 0, 0); // Переміщення позиції на початок файлу
    ved *a=new ved[k]; // Оголошення динамічного масиву
    while (fread(&v, sizeof(ved), 1, f))
    { a[i]=v; i++; } // Записування даних із файлу до масиву
    fclose(f);
    do
    { pr=true; // Припустімо, що масив є відсортовано
      for (i=0; i<k-1; i++)
      { if (a[i].oc < a[i+1].oc) // Якщо сусідні елементи розташовано
        { ved tmp=a[i]; // у неналежному порядку,
          a[i]=a[i+1]; // переставляються елементи
          a[i+1]=tmp;
          pr=false; // і «вимикається» ознака упорядкованості.
        }
      }
    } while(!pr);
    setlocale(0, ".1251");
    if ((f=fopen("ocinki.dat", "wb"))==0) // Створення файлу заново
    { cout<<"Не можу створити файл"<<endl; cin.get(); return; }
    cout<<"Відсортовано за спаданням успішності:"<<endl;
    setlocale(0, ".OCP");
    for (i=0; i<k; i++) // У циклі всі елементи масиву
    { fwrite(&a[i], sizeof(ved), 1, f); // записуються у файл
      for (int j=strlen(a[i].priz); j<15; j++) strcat(a[i].priz, " ");
      cout<<a[i].priz<<" "<<a[i].oc<<endl; // і виводяться на екран
    }
    fclose(f);
    return;
}
```

```

int main()
{
    system("color F0");
    // Визначення імені файлу
    char s[]="ocinki.dat";
    // Виклик функції переглядання вмісту
    // файлу (її код див. у прикладі 12.13)
    look(s);
    sort(s); // Виклик функції сортування
    cin.get(); cin.get();
    return 0;
}

```

Результат виконання
програми →



```

Вміст файла:
Логінова      98
Абрамов       90
Артеха        90
Гулько        90
Балух         89
Чабаненко     74
Качур         65
Кучер         34
Лантінов      25
Іванішин      20
Шкраба        18
Відсортовано за спаданням успішності:
Логінова      98
Абрамов       90
Артеха        90
Гулько        90
Балух         89
Чабаненко     74
Качур         65
Кучер         34
Лантінов      25
Іванішин      20
Шкраба        18

```

Приклад 12.15. Ввести у бінарний файл відомості про сплату комунальних послуг: номер квартири, дата сплати та сума. Визначити, чи сплачено послуги у поточному місяці для певної квартири, номер якої вводиться. Встановити дату і суму останньої сплати за цю квартиру. Створити текстовий файл з відомостями про всі проплати щодо цієї квартири.

Текст програми:

```

#include <iostream>
#include <stdio.h>
#include <ctime>
using namespace std;

struct payment
{
    int apt;
    char date[30];
    float sum;
};
// Записування одного запису у бінарний файл
void add_record(char* name)
{
    payment z; FILE* f;
    f=fopen(name, "ab");
    if(f==NULL) {cout<<"Не вдалось відкрити файл\n"; return;}
    scanf("%d\t%s\t%f", &z.apt, &z.date, &z.sum);
    fwrite (&z,sizeof(payment), 1, f);
    fclose(f);
}

```

```

void view_file(char* name) // Перегляд файлу
{
    payment z; FILE* f;
    f=fopen(name, "rb");
    if (f==NULL) {cout<<"Не вдалось відкрити файл\n"; return;}
    cout<<"\nПерегляд файлу"<<endl;
    cout<<"Квартира\tДата\tСума\n"<<endl;
    while (fread(&z,sizeof(payment), 1, f))
    {
        printf ("%d\t%s\t%10.2f\n", z.apr, z.date, z.sum);
    }
    fclose(f);
}

// Відомості про сплати щодо введеної квартири
bool current_month(char* name, int kv)
{
    payment z; FILE* f;
    bool pr=false;
    time_t t=time(NULL), t1;
    struct tm *tmstr, pmtstr;
    tmstr=localtime(&t);
    int cur_mon=tmstr->tm_mon+1;
    f=fopen(name, "rb");
    if(f==NULL) {cout<<"Не вдалось відкрити файл\n"; return false;}
    while(fread(&z,sizeof(payment), 1, f))
    {
        if (z.apr==kv)
        {
            sscanf(z.date,"%d.%d.%d", &pmtstr.tm_mday, &pmtstr.tm_mon,
                &pmtstr.tm_year);

            if (pmtstr.tm_mon==cur_mon)
                pr=true;
        }
    }
    fclose(f);
    return pr;
}

// Дата останньої сплати щодо введеної квартири
char* last_payment(char* name, int kv)
{
    payment z; FILE* f;
    char *last_date =new char [30]; strcpy(last_date, "");
    time_t t, maxt=0;
    struct tm tmstr, *laststr;
    f=fopen(name, "rb");
    if(f==NULL) {cout<<"Не вдалось відкрити файл\n"; return false;}
    while(fread(&z,sizeof(payment), 1, f))
    {
        if (z.apr==kv)
        {
            sscanf(z.date,"%d.%d.%d", &tmstr.tm_mday, &tmstr.tm_mon,
                &tmstr.tm_year);

```



```

        tmstr.tm_mon-=1; tmstr.tm_year-=1900;
        tmstr.tm_hour=0; tmstr.tm_min=0; tmstr.tm_sec=0;
        t=mktime (&tmstr);
        if (t>maxt) maxt=t;
    }
}
fclose(f);
if (maxt>0)
{ laststr=localtime(&maxt);
  strftime (last_date, 30, "%d/%m/%Y", laststr );
}
return last_date;
}
// Створення текстового файлу
void text_file(char* name, char* tname, int kv)
{ payment z; FILE* f, *ft;
  f=fopen(name, "rb");
  if(f==NULL) {cout<<"Не вдалось відкрити файл\n"; return;}
  ft=fopen(tname, "wt");
  if(ft==NULL) {cout<<" Не вдалось створити файл\n"; return;}
  while(fread(&z,sizeof(payment), 1, f))
  {
      if (z.apr==kv)
          fprintf(ft, "%d\t%s\t%f\n", z.apr, z.date, z.sum);
  }
  fclose(f);fclose(ft);
}
// Перегляд текстового файлу
void view_text_file(char* name)
{ payment z; FILE* f;
  f=fopen(name, "rt");
  if(f==NULL) {cout<<"Не вдалось відкрити файл\n"; return;}
  cout<<"\nПерегляд текстового файлу"<<endl;
  cout<<"Квартира\tДата\tСума\n"<<endl;
  while(!feof(f))
  { fscanf (f, "%d\t%s\t%f\n", &z.apr, z.date, &z.sum);
    printf ("%d\t%s\t%10.2f\n", z.apr, z.date, z.sum);
  }
  fclose(f);
}
int main()
{ system("color F0");
  char* name="payment.txt"; char* tname="txtfile.txt"; //імена файлів
  setlocale(0, ".1251");
  int kv;
  cout<<"Введіть дані про сплату:"<<endl;
  cout<<"Квартира\tДата\tСума\n"<<endl;
  for(int i=0; i<5; i++) add_record(name);
}

```

```

view_file(name);
cout<<"Введіть номер квартири:"<<endl; cin>>kv;
bool oplata = current_month(name, kv);
if (opлата)
    cout<<"У поточному місяці сплачено"<<endl;
else
    cout<<"У поточному місяці не сплачено"<<endl;
cout<<"\nОстання сплата квартири № "<<kv<<endl;
puts(last_payment(name, kv));
text_file(name, tname, kv);
view_text_file(tname);
system("pause");
return 0;
}

```

Результати виконання програми:

Введіть дані про сплату:

Квартира	Дата	Сума
22	03.03.2018	900
1	15.09.2018	1100
4	22.09.2018	250
13	16.05.2018	330
4	29.09.2018	120

Перегляд файлу

Квартира	Дата	Сума
22	03.03.2018	900,00
1	15.09.2018	1100,00
4	22.09.2018	250,00
13	16.05.2018	330,00
4	29.09.2018	120,00

Введіть номер квартири:

4

У поточному місяці сплачено

Остання сплата квартири № 4

29/09/2018

Перегляд текстового файлу

Квартира	Дата	Сума
4	22.09.2018	250,00
4	29.09.2018	120,00

Введіть дані про сплату:

Квартира	Дата	Сума
2	11.08.2018	25,90
4	12.09.2018	190,50
13	15.09.2018	500
23	21.09.2018	80
32	21.09.2018	700

Перегляд файлу

Квартира	Дата	Сума
22	03.03.2018	900,00
1	15.09.2018	1100,00
4	22.09.2018	250,00
13	16.05.2018	330,00
4	29.09.2018	120,00
1	11.09.2018	100,00
2	11.09.2018	822,00
13	12.09.2018	220,00
2	11.08.2018	25,90
4	12.09.2018	190,50
13	15.09.2018	500,00
23	21.09.2018	80,00
32	21.09.2018	700,00

Введіть номер квартири:

13

У поточному місяці сплачено

Остання сплата квартири № 13

15/09/2018

Перегляд текстового файлу

Квартира	Дата	Сума
13	16.05.2018	330,00
13	12.09.2018	220,00
13	15.09.2018	500,00

12.3.2 Робота з бінарними файловими потоками у стилі C++

Для роботи з бінарним форматом файлу використовується прапорець `ios::binary`. Ці файли відкриваються здебільшого в режимі зчитування-записування. Для цього зручними є об'єкти класу `fstream`. Дані у бінарні файли записуються за допомогою методу `write()` класу `ofstream`, а зчитуються – за допомогою методу `read()` класу `ifstream`.

Наприклад:

```

fout.write((char*)&A, sizeof(A));
fin.read((char*)&A, sizeof(A));

```

Перший параметр у цих функцій – адреса змінної `A`. Ці функції працюють з байтами (типу `char`). Для них немає жодного значення, в який спосіб організовано й що собою являють дані, – вони просто переносять їх побайтово з буфера до файлу та навпаки. Другий параметр – довжина змінної `A` у байтах. Тому перший оператор обчислює значення виразу `sizeof(A)` й копіює до файлу, пов'язаного з об'єктом `fout`, обчислену кількість байтів, розпочинаючи з вказаної адреси. Другий оператор копіює кількість байтів `sizeof(A)` з файлу `fin` до змінної `A`.

У бінарному файлі також існує можливість довільного доступу до файлу за допомогою методів `seekp()` та `seekg()`.

При повторному відкритті файлу з використанням тієї ж самої файлової змінної потік слід очищувати за допомогою методу `clear()`, наприклад: `f.clear()`.

Приклад 12.16. Створити файл з інформацією про успішність студентів: прізвище, ім'я, оцінки з інформатики, математики та фізики. Вивести на екран середній бал кожного студента. Виявити, чи є серед студентів збіг прізвищ (однофамільці).

Текст програми:

```
#include <fstream>
#include <iostream>
using namespace std;
fstream f;
struct student
{ char name[15], surname[15];
  int inf, mat, fiz;
} A;
void writing(char* fname)
{ // Створити чи відкрити файл
  f.open(fname, ios::out|ios::app|ios::binary);
  if (!f)
  { cout<<"Не вдається відкрити файл"<<endl; cin.get(); return; }
  f.seekp(0, ios::end); // Позиціонування на кінець файлу
  char yn; // Ознака бажання дописувати нові рядки у файл
rep:
  setlocale(0, ".1251");
  cout<<"Будете вводити дані у файл? y/n"<<endl;
  cin>>yn;
  if (yn!='y') { f.close(); return; }
  cout<<"Введіть прізвище, ім'я, оцінки з інформатики, математики
        та фізики через пробіл"<<endl;
  setlocale(0, ".OCP");
  cin>>A.name>>A.surname>>A.inf>>A.mat>>A.fiz;
  f.write((char*)&A, sizeof(student)); // Записування даних у файл
  goto rep;
}
```

```
void look(char* fname)
{ setlocale(0, ".1251");
  f.open(fname, ios::out|ios::in|ios::binary);
  f.clear();
  if (!f)
  { cout<<"Не вдається відкрити файл"<<endl; cin.get(); return; }
  cout<<"\nВміст файлу:\n";
  setlocale(0, ".OCP");
  while (f.read((char*)&A, sizeof(A)))
  { for (int i=strlen(A.name);i<14; i++) strcat(A.name, " ");
    for (int i=strlen(A.surname);i<14; i++) strcat(A.surname, " ");
    cout << A.name << "\t" << A.surname << "\t" << A.inf << "\t"
      << A.mat << "\t" << A.fiz << endl;
  }
  f.close();
  return ;
}

void namesakes (char* fname)
{ setlocale(0, ".1251");
  f.open(fname, ios::out|ios::in|ios::binary);
  f.clear();
  if (!f)
  { cout<<"Не вдається відкрити файл"<<endl; cin.get(); return; }
  long int k=0;   student A1;
  cout<<"\n361г прізвищ - ";
  bool zbig=false;
  setlocale(0, ".OCP");
  while (f.read((char*)&A, sizeof(A)))
  { while (f.read((char*)&A1, sizeof(A1)))
    { if (strcmp(A.name, A1.name)==0)
      { cout<<A.name<<"\t";
        zbig=true;
      }
    }
    f.clear();
    k++;
    // Позиціонування на структуру файлу з номером k
    f.seekg(k*sizeof(A), ios::beg);
  }
  setlocale(0, ".1251");
  if (!zbig) cout<<"не виявлено";
  cout<<endl;
  f.close();
  return ;
}

int main()
{ system("color F0");
  char s[]="student.dat";      // визначення імені файлу
```

```

writing(s);
look(s);
namesakes(s);
cin.get(); cin.get();
return 0;
}

```

Результати виконання програми:

```

Будете вводити дані у файл? y/n
y
Введіть прізвище, ім'я, оцінки з інформатики, математики та фізики через пробіл
Горбанюк Олена 78 80 85
Будете вводити дані у файл? y/n
n

Вміст файлу:
Іванішин      Олег      56      45      60
Коновець     Олег      89      90      100
Антоненко    Олександр 100     100     98
Борисов      Антон     60      80      74
Горбанюк     Олена     78      80      85

Збіг прізвищ - не виявлено

```

```

Будете вводити дані у файл? y/n
y
Введіть прізвище, ім'я, оцінки з інформатики, математики та фізики через пробіл
Антоненко Ярослава 78 60 95
Будете вводити дані у файл? y/n
n

Вміст файлу:
Іванішин      Олег      56      45      60
Коновець     Олег      89      90      100
Антоненко    Олександр 100     100     98
Борисов      Антон     60      80      74
Горбанюк     Олена     78      80      85
Коновець     Марина     60      82      65
Лич          Олег      45      60      60
Швець       Павло     75      65      80
Антоненко    Ярослава  78      60      95

Збіг прізвищ - Коновець Антоненко

```

12.3.3 Опрацювання бінарних файлів засобами простору імен System::IO платформи .NET Framework

Деякі засоби простору System::IO для роботи з бінарними файлами

Для роботи з бінарними файлами в просторі імен System::IO платформи .NET Framework існують класи File, BinaryReader та BinaryWriter, які дозволять створити файл та здійснювати операції зчитування/записування двійкових даних з/у файл як потік даних.

Розглянемо найпоширеніші методи цих класів.

Поширені дії при роботі з файлами

Дії	Методи
Відкривання файлу у заданому режимі і доступі	<code>File::Open</code>
Записування у файл	<code>Write</code> – записує у потік двійкове подання аргументу
Зчитування з файлу	<code>Read</code> – зчитує з потоку значення аргументу в дужках
Відкриття файлу	<code>File::Open</code>
Закриття файлу (поток)	<code>Close</code>
Переміщення позиції файлу на <code>offset</code> байтів від позиції <code>SeekOrigin::Begin</code> або <code>SeekOrigin::End</code>	<code>BinaryWriter::Seek(offset, SeekOrigin)</code> Наприклад: <code>f->BaseStream->Seek(0, SeekOrigin::Begin);</code> <i>//на початок файлу</i> <code>f->BaseStream->Seek(0, SeekOrigin::End);</code> <i>//у кінець файлу</i>

Наприклад, створити бінарний файл з ім'ям `Test.dat` у поточному каталозі можна командою:

```
BinaryWriter^ f=gcnew BinaryWriter(File::Open("Test.dat",FileMode::OpenOrCreate));
```

або двома командами:

```
FileStream^ data = gcnew FileStream("Test.dat", FileMode::OpenOrCreate);
BinaryWriter^ fb = gcnew BinaryWriter(data);
```

При цьому режим відкриття `OpenOrCreate` дозволить не обнуляти вже наявний файл (якщо такий є), а лише відкрити його.

Якщо створюваний файл слід розмістити не в поточному каталозі, а в якомусь іншому, слід зазначити повний шлях до файлу.

Приклад 12.17. Створити програму, яка дозволить зберігати у бінарному файлі двійкові дані: номер мобільного телефону, дату підключення (активації) номера, вартість однієї хвилини розмов, назву тарифу, назву мобільного оператора. Крім програмного створення файлу і заповнення його даними, передбачити програмний перегляд даних зі створеного файлу. Відібрати абонентів, які користуються мобільним зв'язком понад 10 років без змінення номера телефону.

Текст програми:

```
using namespace System;
using namespace System::IO;

void Zapis(String^ fname)
{
    BinaryWriter^ fb=gcnew BinaryWriter(File::Open(fname,FileMode::OpenOrCreate));
    fb->Seek(0, SeekOrigin::End);
    try
    {
        String ^номер, ^тариф, ^оператор; double плата; DateTime d;
        do
        { Console::Write("\nВведіть номер телефона ");
          номер = Console::ReadLine();
```

```

    if (номер != "")
    { Console::Write("Введіть дату підключення номера - ");
      d = Convert::ToDateTime(Console::ReadLine());
      Console::Write("Введіть вартість 1 хв. розмов - ");
      плата = Convert::ToDouble(Console::ReadLine());
      Console::Write("Введіть назву тарифу - ");
      тариф = Console::ReadLine();
      Console::Write("Введіть оператора - ");
      оператор = Console::ReadLine();
      fb->Write(номер); fb->Write(d.Ticks); fb->Write(тариф);
      fb->Write(плата); fb->Write(оператор);
    }
  } while (номер != "");
} finally { fb->Close(); }
}

void Prosmotr(String^ fname)
{
  if(!File::Exists(fname)) { Console::WriteLine("Файл не знайдено!"); return;}
  BinaryReader^ fb = gcnew BinaryReader(File::OpenRead(fname));
  try
  { Console::WriteLine("\nПерегляд файлу:");
    while (fb->PeekChar() > 0)
    {
      String^ номер = fb->ReadString();
      DateTime d = DateTime( fb->ReadInt64() );
      String^ тариф = fb->ReadString();
      double плата = fb->ReadDouble();
      String^ оператор = fb->ReadString();
      Console::WriteLine(номер + "\t " + d.ToString("d") + "\t " + тариф + "\t " +
        Convert::ToString(String::Format("{0:0.00}",плата)) + "\t " + оператор);
    }
  }
  finally { fb->Close(); }
}

void ThisYear(String^ fname)
{ if(!File::Exists(fname)) { Console::WriteLine("Файл не знайдено!"); return;}
  BinaryReader^ fb = gcnew BinaryReader(File::OpenRead(fname));
  try
  { Console::WriteLine("\nАбоненти, піключені понад 10 років тому:");
    while (fb->BaseStream->Position < fb->BaseStream->Length)
    { String^ номер = fb->ReadString();
      DateTime d = DateTime( fb->ReadInt64() );
      String^ тариф = fb->ReadString();
      double плата = fb->ReadDouble();
      String^ оператор = fb->ReadString();
      if ( d.Year+10 <= DateTime::Now.Year )
        Console::WriteLine(номер + "\t " + d.ToString("d") + "\t " + тариф + "\t " +

```



```

        Convert::ToString(String::Format("{0:0.00}", плата)) + "\t " + оператор);
    } }
    finally { fb->Close(); }
}

int main(array<System::String^> ^args)
{ String^ fname = "telephons.dat";
  Console::WriteLine(L"Введіть дані, які будуть записані у файл.\n
                     Наприкінці двічі натисніть Enter.");

  Zapis(fname);
  Prosmotr(fname);
  ThisYear(fname);
  Console::ReadLine();
  return 0;
}

```

Результат виконання програми:

```

Введіть дані, які будуть записані у файл.
Наприкінці двічі натисніть Enter.

Введіть номер телефона 063-900-80-50
Введіть дату підключення номера - 20.09.2018
Введіть вартість 1 хв. розмов - 0,75
Введіть назву тарифу - Super NET
Введіть оператора - LifeCell

Введіть номер телефона

Перегляд файлу:
067-800-12-34    21.04.2000    Вільний день    0,45    Київстар
050-266-55-40    30.03.1999    3D ноль        0,50    МТС Україна
063-777-54-32    12.03.2015    Смартфон+      0,35    LifeCell
068-70-444-17    29.09.2012    Для смартфона  0,75    Київстар
094-33-79-004    21.09.2011    Non stop       0,55    Інтертелеком
097-977-34-34    25.04.2008    Вільний день    0,45    Київстар
063-900-80-50    20.09.2018    Super NET      0,75    LifeCell

Абоненти, підключені понад 10 років тому:
067-800-12-34    21.04.2000    Вільний день    0,45    Київстар
050-266-55-40    30.03.1999    3D ноль        0,50    МТС Україна
097-977-34-34    25.04.2008    Вільний день    0,45    Київстар

```

Приклад 12.18. Створити програму, яка дозволить зберігати у бінарному файлі двійкові дані: номер мобільного телефону, назву тарифу, назву мобільного оператора, наявність абонплати, вартість однієї хвилини розмов. Крім програмного створення файлу і заповнення його даними, передбачити програмний перегляд даних зі створеного файлу. Відібрати дані про номери телефонів на Київстарі та обчислити їхню кількість і середню вартість 1 хв. розмов. Відібрати тарифи без абонплати. Визначити найдешевший тариф без абонплати.

Розв'язок. Для налаштування форми на ній слід розмістити відповідні командні кнопки і таблиці (елементи `dataGridView`) для введення-виведення даних файлу. Розглянемо докладніше налаштування першого `dataGridView`. Після розмі-

щення його на формі слід по чергово створити на ньому п'ять стовпців за кількістю полів у завданні і при їх створенні відразу задати тип даних і заголовки стовпців:

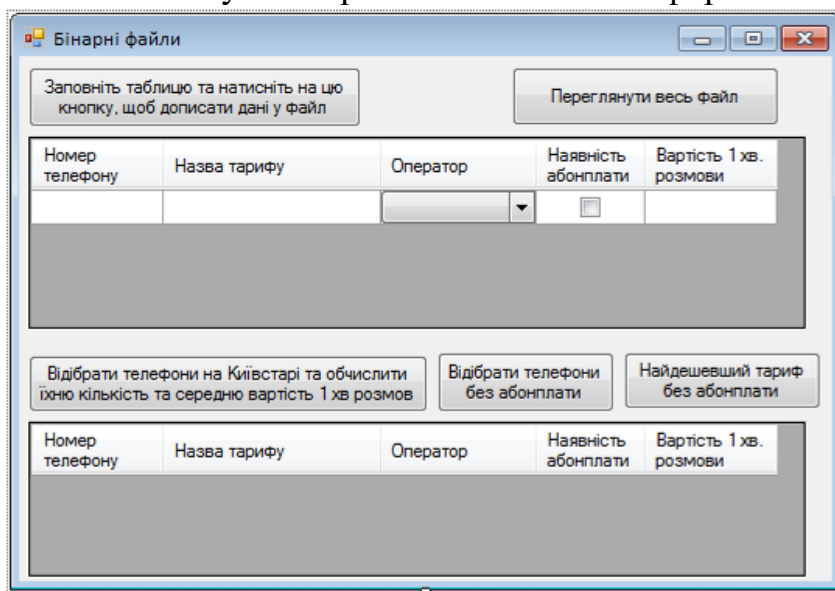
- для 1-го стовпця – у властивості `HeaderText` вписати Номер телефону;
- для 2-го стовпця – у властивості `HeaderText` вписати Назва тарифу;
- для 3-го стовпця – у властивості `HeaderText` вписати Оператор, а для властивості `ColumnType` замість значення за промовчанням вибрати зі списку значення `DataGridViewComboBoxColumn`. Після цього у властивості `Items` (`Items` (Коллекция) ) вписати через *Enter* назви операторів, наприклад: МТС, Київстар, Life;

- для 4-го стовпця – у властивості `HeaderText` вписати Наявність абонплати, а для властивості `ColumnType` замість значення за промовчанням вибрати зі списку значення `DataGridViewCheckBoxColumn`;

- для 5-го стовпця – у властивості `HeaderText` вписати Вартість 1 хв. розмови. Можна також для кожного зі стовпців задати ширину у властивості `Width`.

Тепер можна скопіювати і вставити `dataGridView1`, створивши таким чином вже налаштований `dataGridView2`, для якого лише залишилось для властивостей `Visible` та `AllowUserToAddRows` задати значення `false`.

Після налаштування решти елементів на формі вона набуде вигляду:



Специфікою файлу `Form1.h` з програмним кодом є наявність значної кількості рядків з командами налаштування елементів на формі, які автоматично створюються у цьому файлі при розміщенні нових елементів на формі та встановленні їх властивостей. Тому, щоби вписати команду для зазначення використання простору імен `using namespace System::IO`; з програмними засобами файлового введення-виведення, слід піднятися на початок файлу, віднайти рядок `using namespace System`; і під ним вписати зазначений. Після цього спустися до низу і повернутися до створення програмного коду для п'ятих кнопок `button`.

Текст програми:

```

. . . . .
using namespace System;
using namespace System::IO;
. . . . .

```

```
String ^fname; // Глобальна змінна - ім'я файлу

private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e)
{
    fname = "connect.dat";
}
// Записати дані у файл
private: System::Void button1_Click(System::Object^sender, System::EventArgs^e)
{
    BinaryWriter^ fb=gcnew BinaryWriter(File::Open(fname, FileMode::OpenOrCreate));
    // або замість цієї команди можна скористатися двома такими
    // FileStream^ phons = gcnew FileStream(fname, FileMode::OpenOrCreate);
    // BinaryWriter^ fb = gcnew BinaryWriter(phons);
    fb->Seek(0, SeekOrigin::End); // перейти у кінець файлу для дописування даних
    try
    {
        int k=dataGridView1->Rows->GetRowCount(DataGridViewElementStates::Visible);
        for (int i=0; i < k-1; i++)
        { String ^номер = Convert::ToString(dataGridView1[0,i]->Value);
          // звертатися до клітинки таблиці можна і так: dataGridView1->Rows[i]->Cells[0]->Value
          String ^тариф = Convert::ToString(dataGridView1[1,i]->Value);
          String ^оператор = Convert::ToString(dataGridView1[2,i]->Value);
          bool абонплата = Convert::ToBoolean(dataGridView1[3,i]->Value);
          double var = Convert::ToDouble(dataGridView1[4,i]->Value);
          fb->Write(номер);      fb->Write(тариф);  fb->Write(оператор);
          fb->Write(абонплата);  fb->Write(var);
        }
        dataGridView1->Rows->Clear();
    }
    finally
    { fb->Close(); }
}
// Вивести бінарний файл
private: System::Void button2_Click(System::Object^sender, System::EventArgs^e)
{
    dataGridView1->Rows->Clear();
    if ( File::Exists(fname) == false )
    { MessageBox::Show("file not exists"); return; }
    BinaryReader^ fb = gcnew BinaryReader(File::OpenRead(fname));
    try
    {
        while (fb->PeekChar() > 0)
        {
            String^ номер = fb->ReadString();
            String^ тариф = fb->ReadString();
            String^ оператор = fb->ReadString();
            bool абонплата = fb->ReadBoolean();
            double var = fb->ReadDouble();
        }
    }
}
```

```

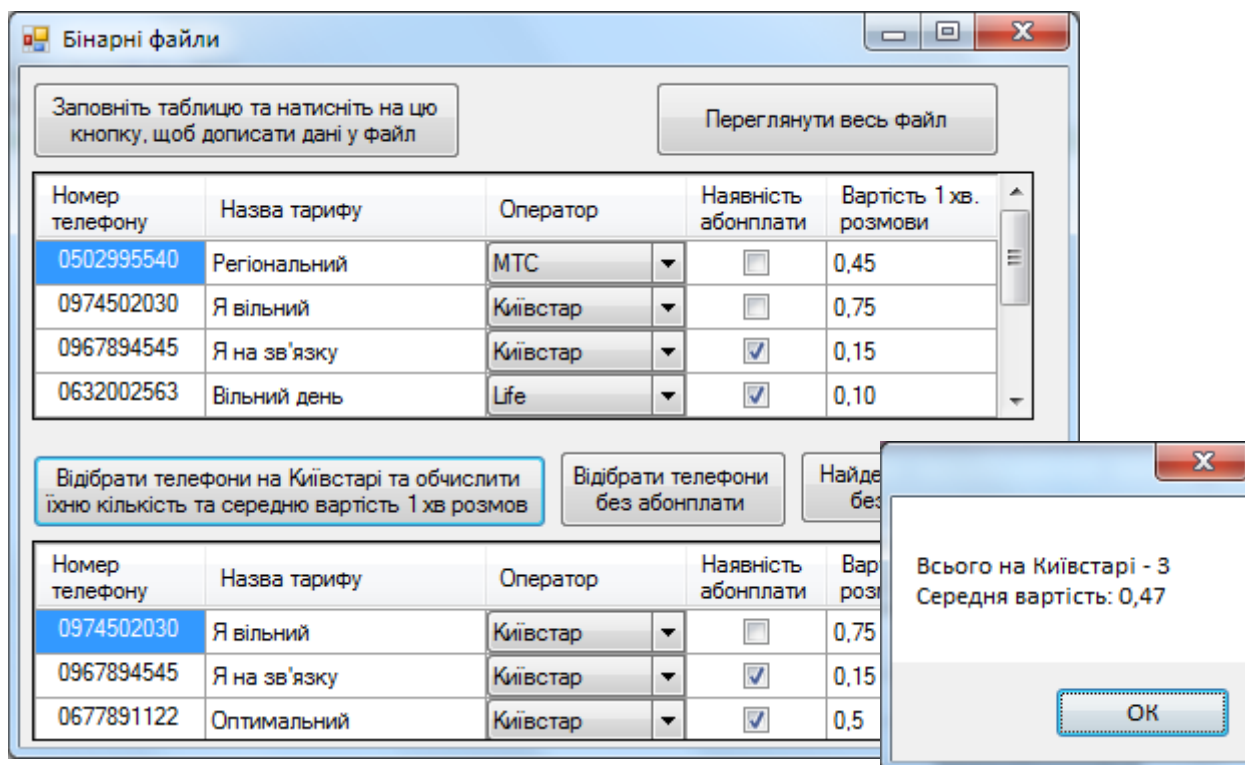
        dataGridView1->Rows->Add(номер, тариф, оператор, абонплата,
                                String::Format("{0:0.00}", var));
    }
}
finally
{ fb->Close(); }
}
// Відібрати телефони на Київстарі та обчислити їх кількість і середню вартість розмов
private: System::Void button3_Click(System::Object^sender, System::EventArgs^e)
{
    int k=0; double s=0;
    dataGridView2->Visible=true; // Відобразити таблицю
    dataGridView2->Rows->Clear(); // Очистити таблицю
    if ( File::Exists(fname) == false )
    { MessageBox::Show("file not exists"); return; }
    BinaryReader^ fb = gcnew BinaryReader(File::OpenRead(fname));
    try
    {
        while (fb->PeekChar() > 0)
        {
            String^ номер = fb->ReadString();
            String^ тариф = fb->ReadString();
            String^ оператор = fb->ReadString();
            bool абонплата = fb->ReadBoolean();
            double var = fb->ReadDouble();
            if (оператор == "Київстар")
            { dataGridView2->Rows->Add(номер, тариф, оператор, абонплата,
                                    String::Format("{0:0.00}", var));

                s += var;
                k++;
            }
        }
        if (k) s /= k;
        MessageBox::Show("Всього на Київстарі - "+(k).ToString()+
                        "\nСередня вартість: "+Convert::ToString(String::Format("{0:0.00}", s)));
    }
    finally
    { fb->Close(); }
}
// Телефони без абонплати
private: System::Void button4_Click(System::Object^sender, System::EventArgs^e)
{
    dataGridView2->Visible=true; // Відобразити таблицю
    dataGridView2->Rows->Clear(); // Очистити таблицю
    if ( File::Exists(fname) == false )
    { MessageBox::Show("file not exists"); return;}
    BinaryReader^ fb = gcnew BinaryReader(File::OpenRead(fname));
    try

```

```
{
    while (fb->PeekChar() > 0)
    {
        String^ номер = fb->ReadString();
        String^ тариф = fb->ReadString();
        String^ оператор = fb->ReadString();
        bool абонплата = fb->ReadBoolean();
        double var = fb->ReadDouble();
        if (!абонплата) dataGridView2->Rows->Add(номер, тариф, оператор,
                                                абонплата, String::Format("{0:0.00}",var));
    }
}
finally
{ fb->Close(); }
}
// Найдешевший тариф без абонплаты
private: System::Void button5_Click(System::Object^sender, System::EventArgs^e)
{
    dataGridView2->Visible = true;
    dataGridView2->Rows->Clear();
    if ( File::Exists(fname) == false )
    { MessageBox::Show("file not exists"); return; }
    BinaryReader^ fb = gcnew BinaryReader(File::OpenRead(fname));
    try
    {
        double var, min=100; // нереальне число
        String ^номер, ^тариф, ^оператор, ^nommin, ^tarmin, ^opmin;
        while (fb->PeekChar() > 0)
        {
            номер = fb->ReadString();
            тариф = fb->ReadString();
            оператор = fb->ReadString();
            bool абонплата = fb->ReadBoolean();
            var = fb->ReadDouble();
            if (!абонплата && min>var)
            { min = var; nommin = номер;
              tarmin = тариф; opmin = оператор;
            }
        }
        dataGridView2->Rows->Add(nommin, tarmin, opmin, false, min);
    }
    finally
    { fb->Close(); }
}
```

Результат виконання програми:



Питання та завдання для самоконтролю

- 1) Що таке файл?
- 2) Яким чином здійснюється програмний доступ до даних файлу?
- 3) Чим схожі і чим відрізняються файли і масиви?
- 4) Який файл називають текстовим?
- 5) Який файл називають бінарним?
- 6) У чому полягає відмінність між бінарними і текстовими файлами?
- 7) Навести засоби роботи з файлами у Visual C++.
- 8) У чому полягає специфіка програмного опрацювання файлів у стилі C?
- 9) В якому заголовному файлі означено тип FILE*?
- 10) Що відбувається при відкриванні файлу функцією fopen()?
- 11) Назвати режими відкривання файлів у стилі C.
- 12) В який спосіб задається режим відкривання бінарного файлу у стилі C?
- 13) Назвати функції файлового форматованого записування-зчитування даних у стилі C.
- 14) Записати C-функцію створення текстового файлу.
- 15) Якою C-функцією можна визначити розмір даних у файлі?
- 16) Навести фрагмент програми для дописування знаку оклику до першого рядка текстового файлу типу FILE*.
- 17) У чому полягає специфіка програмного опрацювання файлів у стилі C++?
- 18) Записати операції «помістити в потік» та «взяти з потоку».

- 19) Які режими відкривання файлів у стилі C++ використовуються для дописування даних у кінець файлу?
- 20) Навести фрагмент коду для відкривання текстового файлу типу FILE* і записування у його кінець структури z зі збереженням формату даних:
`struct LITO { int sun; float sea; }z;`
- 21) Навести команди для зчитування першого і останнього рядка текстового файлу типу FILE*, кожен рядок якого містить два числа (ціле та дійсне) через пробіл, у змінну z типу LITO, оголошену у завданні 23.
- 22) Навести фрагмент коду, який збільшує вдвічі всі значення поля sun змінної z для файлу, зазначеного у завданні 23.
- 23) Навести фрагмент коду, який з одного текстового файлу типу FILE* створює два: у перший записує рядки, які розпочинаються з великої літери, а у другий – решту.
- 24) Навести фрагмент коду, який записує елементи дійсної матриці 3×5 у текстовий файловий потік у стилі C++.
- 25) Навести фрагмент коду, який записує елементи дійсного масиву (вектора) у текстовий файловий потік у стилі C++:
 - а) всі елементи в один рядок через кому;
 - б) кожний елемент окремим рядком.
- 26) Навести і порівняти команди для обнуління файлу при опрацюванні у стилі C та C++.
- 27) Записати команду для створення бінарного файлу.
- 28) Записати команду для переміщення на початок бінарного файлу і ще одну команду для переміщення у кінець цього файлу. В яких випадках ці команди будуть доречні для використання?
- 29) Навести фрагмент коду для зчитування і виведення до елемента rich-TextVox вмісту текстового файлу, кожен рядок якого містить два числа (ціле та дійсне) через пробіл.
- 30) Для чого використовуються методи LoadFile та SaveFile?

БІБЛІОГРАФІЧНИЙ СПИСОК

- 1) C++. Основи програмування. Теорія та практика: підручник / О.Г. Трофименко, Ю.В. Прокоп, І.Г. Швайко, Л.М. Буката та ін.; за ред. О.Г. Трофименко. Одеса: Фенікс, 2010. 544 с.
- 2) C++. Теорія та практика: навч. посіб. з грифом МОНУ/ О.Г. Трофименко, Ю.В. Прокоп, І.Г. Швайко, Л.М. Буката та ін. ; за ред. О.Г. Трофименко. Одеса: ВЦ ОНАЗ, 2011. 587 с.
- 3) ISO 5807:1985 Information processing // Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts. URL: https://www.saiglobal.com/pdftemp/previews/osh/iso/is_o12345_07-02/t011955e.pdf (data of reference: 29.09.2018).
- 4) String Class. URL: <https://docs.microsoft.com/ru-ru/dotnet/api/system.string?redirectedfrom=MSDN&view=netframework-4.7.2> (Data of reference: 29.09.2018).
- 5) Visual C++. NET : пособие для разработчиков C++ / А. Корера, С. Фрейзер, С. Джентайл и др. ; науч. ред. А. Вендров; пер. с англ. В. Пучкова. М.: Лори, 2003. 398 с.
- 6) Библиотека классов платформы .NET Framework. URL: [http://msdn.microsoft.com/ru-ru/lib_rary/gg145045 \(v=vs.110\).aspx](http://msdn.microsoft.com/ru-ru/lib_rary/gg145045 (v=vs.110).aspx) (дата звернення 7.10.2018).
- 7) Дейтел Х.М., Дейтел П.Дж. Как программировать на C++; пер с англ. М.: ООО "Бином-Пресс", 2008. 1456 с.
- 8) Джосьютис Н. C++. Стандартная библиотека. СПб., 2004. 730 с.
- 9) Динман М.И. C++. Освой на примерах. СПб.: БХВ-Петербург, 2006. 384 с.
- 10) Довбуш Г.Ф., Хомоненко А.Д. Visual C++ на примерах / под ред. проф. А.Д. Хомоненко. СПб.: БХВ-Петербург, 2007. 528 с.
- 11) ДСТУ 2873-94 Системи оброблення інформації. Програмування. Терміни та визначення.
- 12) ДСТУ ISO 5807:2016 Оброблення інформації. Символи та угоди щодо документації стосовно даних, програм та системних блок-схем, схем мережевих програм та схем системних ресурсів (ISO 5807:1985, IDT).
- 13) ДСТУ ISO/IEC 8631:2016 Інформаційні технології. Створення та перетворення програм для їх подання (ISO/IEC 8631:1989, IDT).
- 14) Зиборов В.В. MS Visual C++ 2010 в среде .NET. Библиотека программиста. СПб.: Питер, 2012. 320 с.
- 15) Златопольский Д.М. Сборник задач по программированию. 2-е изд. СПб.: БХВ-Петербург, 2007. 240 с.
- 16) Інтегроване середовище розроблення Visual Studio. URL: <https://msdn.microsoft.com/ru-ru/library/dn762121.aspx> (дата звернення 7.10.2018).
- 17) Керівництво по .NET Framework. URL: <https://docs.microsoft.com/ru-ru/dotnet/framework/index> (дата звернення 7.10.2018).
- 18) Ковалюк Т.В. Основи програмування. К.: ВПВ, 2005. 384 с.
- 19) Круппник А.Б. Изучаем C++. СПб.: Питер, 2004. 251 с.

- 20) Крячков А.В., Сухинина И.В., Томшин В.К. Программирование на С и С++. Практикум : учебник. 2-е изд. М.: Горячая линия – Телеком, 2000. 344 с.
- 21) Культин Н.Б. С/С++ в задачах и примерах. – СПб., 2007. 336 с.
- 22) Лафоре Р. Объектно-ориентированное программирование в С++. 4-е изд. СПб.: Питер, 2003. 928 с.
- 23) Либерти Дж., Джонс Б. Освой самостоятельно С++ за 21 день. 5-е изд. М.: Вильямс, 2009. 784 с.
- 24) Липпман С.Б., Лажойе Ж. Язык программирования С++. Полное руководство. Пер. с англ. СПб.: Невский диалект, 2011. 1105 с.
- 25) Мозговой М.В. Классика программирования: алгоритмы, языки, автоматы, компиляторы. Практический поход. СПб.: Наука и техника, 2006. 320 с.
- 26) Огляд середовища CLR. URL: <https://docs.microsoft.com/ru-ru/dotnet/standard/clr> (дата звернення 7.10.2018).
- 27) Павловская Т. А. С/С++. Программирование на языке высокого уровня. СПб.: Питер, 2002. 464 с.
- 28) Павловская Т.А., Щупак Ю.А. С/С++. Структурное программирование: практикум. СПб.: Питер, 2003. 240 с.
- 29) Прата Ст. Язык программирования С++. Лекции и упражнения: учебник; пер. с англ. М.: Диалектика-Вильямс, 2015. 928 с.
- 30) Страуструп Б. Язык программирования С++. Специальное издание; пер. с англ. М.: ООО "Бином-Пресс", 2005. 1104 с.
- 31) Страуструп Б. Дизайн и эволюция С++; пер. с англ. М. : ДМК Пресс; СПб.: Питер, 2007. 445 с.
- 32) Тип системы С++ (современный С++) // Microsoft. Developer Network. URL: <https://msdn.microsoft.com/ru-ru/library/hh279663.aspx> (дата звернення 7.10.2018).
- 33) Трофименко О.Г., Буката Л.М., Леонов Ю.Г. Алгоритмізація обчислювальних процесів і особливості програмування в С++: метод. посібник. Модуль 4. Ч. 2. Одеса: ВЦ ОНАЗ, 2009. 93 с.
- 34) Трофименко О.Г., Прокоп Ю.В. Створення багатомодульних програмних проєктів для опрацювання даних у файлах засобами Visual С++: метод. вказівки для виконання курсової роботи з дисципліни "Основи програмування". Одеса: ВЦ ОНАЗ ім. О. С. Попова, 2015. 44 с.
- 35) Трофименко О.Г., Прокоп Ю.В., Швайко І.Г., Буката Л.М. Основи програмування. Базові алгоритми: метод. вказівки для лаб. і практ. робіт. Ч. 1. Одеса: ВЦ ОНАЗ ім. О. С. Попова, 2014. 108 с.
- 36) Трофименко О.Г., Прокоп Ю.В., Швайко І.Г., Буката Л.М. Основи програмування. Опрацювання структурованих типів: метод. вказівки для лаб. і практ. робіт. Ч. 2. Одеса: ВЦ ОНАЗ ім. О. С. Попова, 2014. 130 с.
- 37) Трофименко О.Г., Прокоп Ю.В., Швайко І.Г., Буката Л.М. Основи програмування. Програмне опрацювання файлів: метод. вказівки для лаб. і практ. робіт. Ч. 3. Одеса: ВЦ ОНАЗ ім. О. С. Попова, 2015. 80 с.
- 38) Хортон А. Visual С++ 2010: полный курс.; пер. с англ. М.:, 2011. 1216 с.
- 39) Шилдт Г. С++: базовый курс. 3-е изд. М.: Вильямс, 2009. 624 с.
- 40) Шилдт Г. Справочник программиста по С/С++. М., 2006. 432 с.

ДОДАТОК А

ТАБЛИЦІ КОДІВ ASCII

Таблиця А.1

Символи ASCII

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	NUL 0	SOH 1	STX 2	ETX 3	EOT 4	ENQ 5	ACK 6	BEL 7	BS 8	TAB 9	LF 10	VT 11	FF 12	CR 13	SO 14	SI 15
1.	DLE 16	DC1 17	DC2 18	DC3 19	DC4 20	NAK 21	SYN 22	ETB 23	CAN 24	EM 25	SUB 26	ESC 27	FS 28	GS 29	RS 30	US 31
2.	(sp) 32	! 33	" 34	# 35	\$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
3.	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
4.	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
5.	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
6.	` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
7.	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	 124	} 125	~ 126	DEL 127

Таблиця А.2

Керувальні символи ASCII

Код	Ім'я	Ctrl-код	Призначення
0	NUL (Null – порожньо, дані є відсутні)	^@	Використовується для передавання у разі відсутності даних. Вживається у багатьох мовах програмування як кінець рядка. В деяких операційних системах NUL – останній символ кожного текстового файлу
1	SOH (Start of Heading – початок заголовка)	^A	Використовується для зазначення початку заголовка, який може містити інформацію про маршрутизацію чи адресу
2	STX (Start of Text – початок тексту)	^B	Вказує на початок тексту і водночас на кінець заголовка
3	ETX (End of Text – кінець тексту)	^C	Використовується при завершенні тексту, який розпочинався символом STX
4	EOT (End of Transmission – кінець передачі)	^D	У системі UNIX [Ctrl]+[D] означає кінець файлу при введенні даних з клавіатури
5	ENQ (Enquiry – запит)	^E	Запит ідентифікаційних даних від віддаленої станції

Продовження табл. А.2

Код	Ім'я	Ctrl-код	Призначення
6	ACK (Acknowledge – потвердження)	^F	Приймальний пристрій передає цей символ відправникові як потвердження успішного прийому даних
7	BEL (Bell – дзвоник, звуковий сигнал)	^G	Використовується для керування пристроями сигналізації. У мовах програмування C та C++ позначається \a
8	BS (BackSpace – повертання на один символ)	^H	Повертання на одну позицію назад, відбувається вилучання попереднього символу
9	HT (Horizontal Tabulation – горизонтальне табулювання), чи TAB	^I	Відбувається переміщення курсора до наступної позиції табуляції. У багатьох мовах програмування позначається \t
10	LF (Line Feed – переведення рядка)	^J	Відбувається переміщення курсора до початку наступного рядка (на один рядок вниз). У кінці кожного рядка текстового файлу ставиться чи то цей символ, чи CR, чи вони обидва разом (CR, а потім LF), залежно від операційної системи. У багатьох мовах програмування позначається \n і при виведенні тексту призводить до переведення рядка
11	VT (Vertical Tabulation – вертикальне табулювання)	^K	Відбувається переміщення курсора до наступної групи рядків
12	FF (Form Feed – переведення сторінки)	^L	Відбувається переміщення курсора до початку наступної сторінки, форми чи екрана
13	CR (Carriage Return – переведення каретки)	^M	Відбувається переміщення курсора до початку (крайньої лівої) позиції поточного рядка. У багатьох мовах програмування позначається \r і використовується для повертання на початок рядка без переведення рядка. У деяких операційних системах цей символ позначається [Ctrl]+[M] і ставиться наприкінці кожного рядка текстового файлу перед LF
14	SO (Shift Out)	^N	Указує, що всі наступні кодові комбінації мають інтерпретуватися згідно із зовнішнім набором символів до приходу символу SI
15	SI (Shift In)	^O	Указує, що наступні кодові комбінації мають інтерпретуватися згідно зі стандартним набором символів

Продовження табл. А.2

Код	Ім'я	Ctrl-код	Призначення
16	DLE (Data Link Escape – перемикання)	^P	Використовується для змінювання значення наступних символів для додаткового контролю чи для передавання довільної комбінації бітів
17	DC1/XON	^Q	(Device Controls – контроль пристрою) символи для керування допоміжними пристроями (спеціальними функціями)
18	DC2	^R	
19	DC3/XOFF	^S	
20	DC4	^T	
21	NAK (Negative Acknowledgement – непотвердження)	^U	Приймальний пристрій передає цей символ відправнику у разі відмови прийому даних
22	SYN (Synchronous/Idle – синхронізація)	^V	Використовується у синхронізованих системах передавання. У моменти відсутності передавання даних система безперервно надсилає символи SYN для забезпечення синхронізації
23	ETB (End of Transmission Block – кінець блока передачі)	^W	Указує на кінець блока даних для комунікаційних цілей. Використовується для розбиття на окремі блоки великих обсягів даних
24	CAN (Cancel – відміна)	^X	Зазначає, що даними, які передували цьому символу у повідомленні чи блоці, слід знехтувати (зазвичай у разі виявлення помилки)
25	EM (End of Medium – кінець носія)	^Y	Указує на фізичний кінець носія інформації
26	SUB (Substitute – заміник)	^Z	Використовується для заміни помилкового чи неприпустимого символу. [CTRL]+[Z] використовується як кінець файлу при введенні даних з клавіатури у системах DOS та Windows
27	ESC (Escape – розширення)	^[	Використовується для розширення коду, зазначаючи те, що наступний символ має альтернативне значення
28	FS (File Separator – розділювач файлів)	^\ ^]	Набір керувальних символів, що використовувались у минулому як розділювачі меж порцій інформації
29	GS (Group Separator – розділювач груп)	^]	
30	RS (Record Separator – розділювач записів)	^^	
31	US (Unit Separator – розділювач юнітів)	^_	

Закінчення табл. А.2

Код	Ім'я	Ctrl-код	Призначення
32	(sp) (Space – пробіл)		Недрукований символ для розділювання слів чи переміщення курсора на екрані вперед на одну позицію
127	DEL (Delete – вилучання)	^?	Використовується для вилучання (стирання) символу з поточної позиції курсора

Таблиця А.3

Кодування другої половини ASCII

Кодування ср866

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	␣	␤	␥	␦	␧	␨	␩	␪	␫	␬	␭	␮	␯	␰	␱	␲
	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	␣	␤	␥	␦	␧	␨	␩	␪	␫	␬	␭	␮	␯	␰	␱	␲
	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	␣	␤	␥	␦	␧	␨	␩	␪	␫	␬	␭	␮	␯	␰	␱	␲
	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	ё	ё	≥	≤	┌	┐	÷	я	°	▪	·	√	а	з	■	□
	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Кодування ср1251 (Windows-1251)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	Ъ	Г	,	і	„	...	†	‡	€	%	Љ	«	Њ	К	Ћ	Ц
	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	ђ	‘	’	”	”	•	–	—		™	љ	»	њ	ќ	ћ	џ
	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A		у	ў	Ј	Ѡ	Г	І	Ѓ	Ё	©	Є	«	¬	·	®	Ї
	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	°	±	І	і	Ѓ	μ	¶	·	ё	№	є	»	ј	Ѕ	ѕ	ї
	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

ДОДАТОК Б

ОПЕРАЦІЇ МОВИ C++

Таблиця Б.1

Операція	Опис	Напрямок
<i>Унарні операції</i>		
:: . -> [] ()	доступ до області видимості вибір елемента через об'єкт вибір елемента через вказівник індекс масиву дужки, виклик функції	⇒
<min>() ++ -- typeid dynamic_cast static_cast reinterpret_cast const_cast sizeof -- ++ ~ ! - + & * new delete (<min>) .* ->*	конструювання постфіксний інкремент постфіксний декремент ідентифікація типу часу виконання перетворення типу з перевіркою на етапі виконання перетворення типу з перевіркою на етапі компіляції перетворення типу без перевірки константне перетворення типу розмір об'єкта чи типу в байтах префіксний декремент префіксний інкремент порозрядне НЕ (заперечення, інверсія) логічне НЕ (заперечення, інверсія) унарний мінус унарний плюс адреса розрадресація динамічне виділення пам'яті динамічне звільнення пам'яті зведення типів вибір на елемент через об'єкт вибір на елемент через вказівник	⇐
<i>Бінарні й тернарна операції</i>		
* / %	множення ділення остача від ділення	⇒
+ -	додавання віднімання	⇒
<< >>	порозрядний зсув ліворуч порозрядний зсув праворуч	⇒
< <= > >=	менше менше чи дорівнює більше більше чи дорівнює	⇒

Закінчення табл. А.2

Операція	Опис	Напрямок
==	дорівнює	⇒
!=	не дорівнює	⇒
&	порозрядна кон'юнкція (І)	⇒
^	порозрядне виключне АБО	⇒
	порозрядна диз'юнкція (АБО)	⇒
&&	логічне І	⇒
	логічне АБО	⇒
? :	умовна операція (тернарна)	⇐
=	присвоювання	⇐
*=	множення з присвоюванням	
/=	ділення з присвоюванням	
%=	остача від ділення з присвоюванням	
+=	додавання з присвоюванням	
-=	віднімання з присвоюванням	
<<=	зсув ліворуч з присвоюванням	
>>=	зсув праворуч з присвоюванням	
&=	порозрядне І з присвоюванням	
=	порозрядне АБО з присвоюванням	
^=	порозрядне виключне АБО з присвоюванням	
throw	генерування виняткової ситуації	
,	кома, послідовне обчислювання	⇒
Операції наведено у порядку зменшення пріоритету. Операції з різними пріоритетами розділені рискою		

ДОДАТОК В

ФУНКЦІЇ СТАНДАРТНОЇ БІБЛІОТЕКИ C++.

ВМІСТ ЗАГОЛОВНИХ ФАЙЛІВ

Таблиця В.1

Заголовний файл <ctype.h> (<cctype>) – функції класифікації і перетворення символів

Функція	Опис
isalnum	Перевіряє, є символ літерою чи цифрою
isalpha	Перевіряє, чи є символ літерою
iscntrl	Перевіряє, чи є символ керувальним
isdigit	Перевіряє, чи є символ цифрою
isgraph	Перевіряє, чи є символ видимим
islower	Перевіряє, чи є символ літерою нижнього регістру
isprint	Перевіряє, чи є символ друкованим
ispunct	Перевіряє, чи є символ пунктуаційним
isspace	Перевіряє, чи є символ розділювальним
isupper	Перевіряє, чи є символ літерою верхнього регістру
iswalnum	Перевіряє, є символ літерою чи цифрою
iswalpha	Перевіряє, чи є символ літерою
iswcntrl	Перевіряє, чи є символ керувальним
iswctype	Перевіряє багатобайтовий символ
iswdigit	Перевіряє, чи є символ цифрою
iswgraph	Перевіряє, чи є символ видимим
iswlower	Перевіряє, чи є символ літерою нижнього регістру
iswprint	Перевіряє, чи є символ друкованим
iswpunct	Перевіряє, чи є символ пунктуаційним
iswspace	Перевіряє, чи є символ розділювальним
iswupper	Перевіряє, чи є символ літерою верхнього регістру
iswxdigit	Перевіряє, чи є символ шістнадцятковою цифрою
isxdigit	Перевіряє, чи є символ шістнадцятковою цифрою
tolower	Повертає символ у нижньому регістрі
toupper	Повертає символ у верхньому регістрі
tolower	Повертає символ у нижньому регістрі
toupper	Повертає символ у верхньому регістрі

Заголовний файл <string.h> (<cstring>) – функції локалізації

localeconv	Повертає опис налагоджень локального середовища
setlocale	Встановлює нову локалізацію

Заголовний файл <math.h> (<cmath>) – математичні функції

Функція	Опис
acos	Повертає арккосинус аргументу
asin	Повертає арксинус аргументу
atan	Повертає арктангенс аргументу
atan2	Повертає арктангенс відношення аргументів
ceil	Округлює його до більшого цілого
cos	Обчислює косинус
cosh	Обчислює гіперболічний косинус
exp	Обчислює експоненту
fabs	Повертає модуль числа
floor	Округлює його до меншого цілого
fmod	Повертає остачу від ділення x на y
frexp	Виокремлює з числа мантису та експоненціальну частину
ldexp	Перетворює мантису та показник степеня на число
log	Обчислює натуральний логарифм
log10	Обчислює десятковий логарифм
modf	Розбиває число на цілу й дійсну частини
pow	Підносить число до степеня
sin	Обчислює синус
sinh	Обчислює гіперболічний синус
sqrt	Обчислює квадратний корінь
tan	Повертає тангенс аргументу
tanh	Повертає гіперболічний тангенс аргументу

Заголовний файл <signal.h> (<csignal>) – функції опрацювання сигналів

raise	Перериває виконання програми
signal	Ресструє опрацювання сигналів

Заголовний файл <stdio.h> (<cstdio>) – функції введення-виведення у стилі C

clearerr	Очищує прапорці помилок при роботі з потоком
fclose	Закриває потік введення-виведення
feof	Перевіряє досягання кінця файлу
terror	Повертає код помилки при роботі з потоком
fflush	Записує дані з буфера
fgetc	Читає з потоку символ
fgetpos	Повертає поточну позицію у файлі
fgets	Читає з потоку в рядок n символів
fgetwc	Читає з потоку символ
fgetws	Читає з потоку в рядок n символів
fopen	Відкриває потік введення-виведення
fprintf	Записує дані до потоку

Заголовний файл <stdio.h> (<cstdio>) – функції введення-виведення у стилі C

Функція	Опис
fputc	Записує символ до потоку
fputs	Записує рядок символів до потоку
fputwc	Записує символ до потоку
fputws	Записує рядок символів до потоку
tread	Зчитує дані з потоку введення
freopen	Відкриває потік введення-виведення
fscanf	Виводить з файлу форматовані дані
fseek	Переміщує позицію у файлі відносно поточної
fsetpos	Переміщує поточну позицію у файлі відносно його початку
ftell	Повертає поточну позицію у файлі
fwprintf	Записує дані до потоку
fwrite	Записує дані із заданого буфера до потоку
fwscanf	Зчитує з потоку дані й записує їх по заданих адресах
getc	Зчитує символ з потоку
getchar	Зчитує символ зі стандартного введення
gets	Зчитує символи з клавіатури до появи символу нового рядка
getwc	Зчитує символ з потоку
getwchar	Повертає черговий символ з клавіатури
perror	Виводить рядок вигляду «s: error повідомлення»
printf	Виводить рядок параметрів у певному форматі
putc	Записує символ до потоку
putchar	Виводить символ на стандартний пристрій виведення
puts	Виводить рядок на стандартний пристрій виведення
putwc	Записує символ до потоку
putwchar	Виводить символ на стандартний пристрій виведення
remove	Вилучає файл
rename	Перейменовує файл
rewind	Очищує прапорці помилок при роботі з потоком і переходить до початку файлу
scanf	Вводить рядок параметрів у певному форматі
setbuf	Встановлює буферизацію потоку введення-виведення
setvbuf	Перетворює рядки на основі поточної локалізації
sprintf	Виводить рядок параметрів у певному форматі
sscanf	Вводить дані з рядка
swprintf	Виводить рядок параметрів у певному форматі
swscanf	Вводить дані з рядка
tmpfile	Відкриває потік двійкового введення-виведення до тимчасового файлу
tmpnam	Створює унікальне ім'я файлу
ungetc	Повертає символ до потоку

Заголовний файл <stdio.h> (<cstdio>) – функції введення-виведення у стилі C

Функція	Опис
ungetc	Повертає символ до потоку
vfprintf	Надсилає відформатоване виведення до потоку
vfwprintf	Надсилає відформатоване виведення до потоку
vprintf	Надсилає відформатоване виведення до стандартного потоку виведення
vsprintf	Виводить рядок параметрів у певному форматі
vswprintf	Виводить рядок параметрів у певному форматі
wprintf	Надсилає відформатоване виведення до стандартного потоку виведення
wscanf	Виводить рядок параметрів у певному форматі
wscanf	Вводить рядок параметрів у певному форматі

Заголовний файл <stdlib.h> (<cstdlib>) – різні функції у стилі C

abort	Перериває виконання програми
abs	Повертає модуль числа
atexit	Реєструє функцію, яка викликається по завершенні роботи програми
atof	Перетворює рядок на дійсне число
atoi	Перетворює рядок на ціле число
atol	Перетворює рядок на довге ціле число
bsearch	Відшукує елемент у відсортованому масиві
calloc	Виділяє блок пам'яті
div	Ділення з остачею
exit	Перериває виконання програми
free	Звільнює блок пам'яті
getenv	Повертає значення змінної оточення
labs	Повертає модуль числа
ldiv	Ділення з остачею
malloc	Виділяє блок пам'яті
mblen	Визначає розмір багатобайтових символів
mbstowcs	Перетворює рядок багатобайтових символів на масив з wchar_t
mbtowc	Перетворює багатобайтовий символ на wchar_t
qsort	Сортує заданий масив
rand	Генерує випадкові числа
realloc	Змінює розміри раніш виділеної пам'яті
srand	Встановлює початкове псевдовипадкове число
strtod	Перетворює рядок на число
strtol	Перетворює рядок на число з урахуванням системи числення
strtoul	Перетворює рядок на число з урахуванням системи числення
system	Передає рядок командному процесорові ОС
wcstod	Перетворює рядок на число

Заголовний файл <stdlib.h> (<cstdlib>) – різні функції у стилі C

Функція	Опис
wcstol	Перетворює рядок в число з урахуванням системи числення
wcstombs	Перетворює wchar_t на рядок багатобайтових символів
wcstoul	Перетворює рядок на число з урахуванням системи числення
wctomb	Перетворює wchar_t на багатобайтовий символ

Заголовний файл <string.h> (<cstring>) – роботи з рядками функції у стилі C

memchr	Відшукує перше входження символу до блока пам'яті
memcmp	Порівнює блоки пам'яті
memcpy	Копіює блок пам'яті
memmove	Переносить блок пам'яті
memset	Заповнює блок пам'яті символом
strcat	Сполучає рядки
strchr	Відшукує символ у рядку
strcmp	Порівнює рядки
strcoll	Порівнює рядки з урахуванням встановленої локалізації
strcpy	Копіює один рядок до іншого
strcspn	Відшукує один із символів одного рядка в іншому
strerror	Повертає вказівник на рядок з описом помилки
strlen	Повертає довжину рядка
strncat	Сполучає один рядок з n символами іншого
strncmp	Порівнює один рядок з n символами іншого
strncpy	Копіює перші n символів одного рядка до іншого
strpbrk	Відшукує один із символів одного рядка в іншому
strrchr	Відшукує символ в рядку
strspn	Відшукує символ одного рядка, який є відсутній в іншому
strstr	Відшукує підрядок у рядку
strtok	Виокремлює з рядка лексеми
strxfrm	Перетворює рядки на основі поточної локалізації
wscat	Сполучає рядки
wcschr	Відшукує символ у рядку
wscmp	Порівнює рядки
wscoll	Порівнює рядки з урахуванням встановленої локалізації
wscpy	Копіює один рядок до іншого
wscspn	Відшукує один із символів одного рядка в іншому
wcslen	Повертає довжину рядка
wcsncat	Сполучає один рядок з n символами іншого
wcsncmp	Порівнює один рядок з n символами іншого
wcsncpy	Копіює перші n символів одного рядка до іншого
wspbrk	Відшукує один із символів одного рядка в іншому
wsrchr	Відшукує символ у рядку

Заголовний файл <string.h> (<cstring>) – роботи з рядками функції у стилі C

Функція	Опис
wcsspn	Відшукує символ одного рядка, відсутній в іншому
wcsstr	Відшукує підрядок у рядку
wcstok	Виокремлює з рядка лексеми
wcstrxfrm	Перетворює рядки на основі поточної локалізації
wmemcpy	Копіює блок пам'яті
wmemmove	Переносить блок пам'яті
wmemset	Заповнює блок пам'яті символом

Заголовний файл <time.h> (<ctime>) – функції для роботи з датою і часом у стилі C

asctime	Перетворює дату-час на рядок
clock	Повертає час виконання програми
ctime	Перетворює час на рядок
difftime	Повертає різницю в часі
gmtime	Ініціалізує структуру tm на основі time_t
localtime	Ініціалізує структуру tm на основі time_t
mktime	Заповнює поля дня тижня і дня року
strftime	Перетворює час у форматі fmt на формат tm
time	Повертає поточні дату/час у вигляді time_t
wcsftime	Повертає час у форматі fmt до формату tm

Заголовний файл <wchar.h> (<cwchar>) – функції для роботи з багатобайтовими символами у стилі C

btowc	Перетворює символ і довге ціле число
fwide	Визначає вигляд потоку
wmemchr	Відшукує перше входження символу до блока пам'яті
wmemcmp	Порівнює блоки пам'яті

ПРЕДМЕТНЫЙ ПОКАЖЧИК

#

#, 341, 342, 344
 ##, 342
 #define, 342
 #elif, 341, 344
 #else, 341, 344
 #endif, 341, 343, 344
 #error, 341, 344
 #if, 341, 344
 #ifdef, 341, 344
 #ifndef, 341, 344
 #include, 36, 339, 342, 359
 #line, 341, 344
 #pragma, 341, 344
 #undef, 343

—

_strdate, 381
 _strtime, 381

A

abs, 61, 467
 acos, 61, 465
 Activated, 27
 Add, 25, 192, 392
 AddDays, 392
 AddHours, 392
 AddMinutes, 392
 AddMonths, 392
 AddYears, 392
 algorithm, 8
 AllowUserToAddRows, 153, 160, 192
 API, 15
 append, 281, 283
 AppendText, 423
 array, 395
 ASCII, 240, 244, 255, 299, 458
 asctime, 380, 383, 469
 asin, 61, 465
 assign, 281, 284
 atan, 61, 465
 atan2, 61, 465
 atof, 277, 279, 417, 467
 atoi, 277, 415, 417, 467
 atol, 277, 467
 auto, 345
 AutoSize, 28, 30

B

BackColor, 28, 30
 BaseStream, 425
 bool, 50, 52, 73
 Boolean, 53
 break, 95, 104, 131, 145, 146
 button, 21, 25, 28, 184
 Byte, 53

C

c_str, 280, 300
 C++ shell, 41
 calloc, 217, 230, 467
 ceil, 61, 465
 Cells, 152
 cerr, 38
 ChangeExtension, 424
 char, 50, 52, 241, 242
 Char, 53, 289, 295
 chart, 109, 111
 calloc, 230
 checkBox, 26
 cin.get, 37
 Codepad, 44
 checkBox, 26, 33, 34
 Checked, 26, 33
 cin, 19, 35, 249
 cin.get, 37
 Clear, 25, 29, 68, 110
 Click, 27
 clock, 380, 469
 clock_t, 380
 Close, 30, 396, 423, 425
 CLR, 39
 Codepad, 44
 Color, 30, 34, 110
 ColumnCount, 192
 ColumnHeadersVisible, 153, 160, 180, 192
 comboBox, 25
 Compare, 291, 392
 const, 54, 55, 66, 67
 Contains, 399
 continue, 104, 146
 Convert, 40
 copy, 281, 284
 CopyTo, 424
 cos, 61, 465

cosh, 61, 465
cout, 35, 36, 37, 38
Create, 423
CreateText, 423
ctime, 381, 469

D

dataGridView, 26, 152, 153, 160
Date, 391
DateTime, 388, 391
DateTimeOffset, 389, 391
DateTimePicker, 393
Day, 392
DayOfWeek, 392
DaysInMonth, 393
default, 95
defined, 342
delete, 216, 218, 230
Delete, 424
difftime, 381, 469
double, 50, 51
Double, 53
DoubleClick, 27
do-while, 102, 130, 132, 133, 138, 141
DragDrop, 27
dynamic link library, 350

E

empty, 281, 285
Enabled, 28
Enter, 27
enum, 50, 52, 376
Equals, 291, 393
ERANGE, 278
erase, 281, 285, 286
errno, 278, 320
Events, 26, 32
Exists, 424
exp, 61, 465
extern, 337, 338, 339, 345

F

fabs, 61, 465
fclose, 404, 407, 465
feof, 404, 414, 465
fgets, 404, 465
FILE, 432
File::Move, 424
find, 281, 284
float, 50, 51
floor, 61, 465

fmod, 465
Font, 25, 27, 28, 30
fopen, 403, 465
for, 102, 132, 137, 141
ForeColor, 31
fprintf, 406
fputs, 405, 414, 466
fread, 432
free, 218, 467
frexp, 465
fscanf, 405, 466
fseek, 406, 434, 466
fstream, 418
ftell, 406, 434, 466
fwrite, 433

G

gcnew, 211
getch, 36, 37, 71, 244
GetExtension, 424
GetFileName, 424
GetFullPath, 424
getline, 249, 301, 302, 420
gets, 152, 249, 267
gets_s, 249, 250
gmtime, 381, 469
goto, 72, 97, 98, 104, 413
groupBox, 33
GroupBox, 228
GUI, 15

H

heap, 216
Height, 30, 319
Hide, 29
Hour, 392
hScrollBar, 33

I

IDE, 15
Ideone, 42
if, 81
if – else, 83
ifstream, 418, 422, 443
Image, 114
IndexOf, 289
inline, 307
InputBox, 226, 228
insert, 281, 287
Insert, 25, 290
int, 50

Int16, 53
Int32, 53
Int64, 53
Interval, 32
isalnum, 243, 464
isalpha, 243, 464
iscntrl, 243
isdigit, 243, 464
IsDigit, 291, 293
isgraph, 243
IsLetter, 291
islower, 243, 464
IsLower, 291
isprint, 243, 464
ispunct, 243, 464
IsPunctuation, 291
isspace, 243, 266, 464
isupper, 243, 277, 464
IsUpper, 291, 294, 426
IsWhiteSpace, 291
Items, 25, 26, 28, 101, 152, 153, 450
itoa, 278, 414, 415

K

KeyDown, 27
KeyPress, 27
KeyUp, 27

L

label, 25, 28
ldexp, 465
Leave, 27
Left, 32
length, 281, 288
Length, 119, 218, 222, 289, 292, 295, 424
Lines, 119
ListBox, 25
Load, 27
localtime, 381, 383, 384, 469
Location, 28
log, 61, 465
log10, 61, 465
long, 50
long double, 50, 51, 53
ltoa, 278

M

M_E, 61
M_PI, 61
main(), 36
malloc, 60, 217, 218

Maximum, 33
menuStrip, 26
MessageBox, 85, 86, 87
Minute, 392
mktime, 381, 469
modf, 61, 465
Month, 392
MonthCalendar, 394
MouseDown, 27
MouseMove, 27, 32
MouseUp, 27
MoveTo, 424
Multiline, 25
mutable, 346

N

Name, 160
namespace, 22, 39, 347, 348
new, 216, 218, 229
Now, 32, 392, 394
NULL, 207, 208, 211, 217, 248

O

ofstream, 418, 422
OnClick, 26
Open, 423
openFileDialog, 430
OpenRead, 423
OpenWrite, 423

P

panel, 33
Parse, 40, 393
Peek, 424
pow, 57, 61, 62, 465
printf, 36, 38, 46, 63, 99, 249
Properties, 22, 26
public, 22
puts, 39, 95, 152, 249

R

RAD, 15
radioButton, 26
RadioButton, 33
rand, 62, 92, 140, 230
random, 62
randomize, 62
read, 443
Read, 39, 423
ReadAllText, 423
ReadLine, 39

realloc, 217, 467
register, 345
Remove, 25, 290
replace, 281, 282
Replace, 290
reserve, 281
resize, 281, 282
return, 255, 305, 306
richTextBox, 25, 430
RowHeadersVisible, 153, 160, 192
Rows, 152

S

saveFileDialog, 430
SByte, 53
scanf, 36, 38, 46, 249
Seek, 424, 425
seekg, 422, 444
seekp, 422, 444
SelectedIndex, 25, 28
setlocale, 36, 250, 464
setprecision, 38
setw, 37, 422
short, 50, 51
Show, 28, 29, 85, 87
signed, 51
sin, 61, 465
Single, 53
sinh, 61, 465
size, 281
Size, 28, 30, 31
sizeof, 53, 365, 444
Sorted, 25, 29
Split, 290
sqrt, 61, 465
srand, 63, 92, 230, 467
static, 227, 311, 338, 346
static library, 350
strcat, 251, 253, 414
strchr, 252, 253
strcmp, 252, 253
strcpy, 252, 253
strcspn, 252, 253
StreamReader, 424
strftime, 381, 384, 469
stricmp, 252
string, 279, 280
String, 288, 289
strlen, 251, 253
strlwr, 252, 254
strncat, 251
strncmp, 252

strncpy, 252, 253, 266, 271
strnicmp, 252
strnset, 253
strpbrk, 253
strrchr, 252, 253
strrev, 253
strset, 253
strspn, 252
strstr, 252, 253
strtod, 278, 467
strtok, 253, 254, 266, 271
strtol, 278, 467
strtoul, 278, 467
struct, 364
strupr, 252, 253
substr, 281
Substring, 289
Subtract, 393
swap, 281
switch, 95
swprintf, 300
swscanf, 300
system, 467

T

tabControl, 25
tan, 61, 465
tanh, 465
tellg, 422
tellp, 422
Text, 21, 22, 25, 28, 30
TextAlign, 30
textBox, 25, 28
TextChanged, 27
Tick, 32
Ticks, 392
time, 381, 469
time_t, 63, 92, 230, 380
timer, 32
Timer, 298
TimeSpan, 389
tm, 380, 383
Today, 392, 395
ToDouble, 40, 152, 222
ToInt32, 40
tolower, 243, 244, 464
ToLower, 290
ToSingle, 40
ToString, 40, 393
toupper, 243, 244, 464
ToUpper, 290
Trim, 290, 295, 428

TrimEnd, 290
TrimStart, 290
typedef, 53, 151, 177, 340, 362
tzset, 381, 384

U

UInt32, 53
UInt64, 53
ultoa, 278
Unicode, 241, 288, 289, 299, 300, 301
union, 363
unsigned, 50, 51
using, 36
using namespace, 19, 23
UTF-16, 241, 289
UTF-32, 241
UTF-8, 241, 423

V

visible, 29
void, 50, 60, 93, 306
volatile, 346

W

Wandbox, 44
wchar_t, 299, 300
wcin, 299, 300
wcout, 300
wcscat, 300
wcschr, 300
wcslen, 299
wcspbrk, 300
while, 102, 130, 132, 137, 141
Width, 30, 33, 153, 160, 192, 450
WordWrap, 25
write, 443
Write, 39, 423
WriteAllText, 423
WriteLine, 39, 423, 425
wstring, 300

Y

Year, 392

A

автоматична змінна, 312, 345
адресація, 207
алгоритм, 8
алфавіт мови, 48
арифметичний вираз, 55
арифметичні операції, 56

Б

бібліотека, 350
бібліотеки функцій, 350
бінарні (двійкові) файли, 402
блок, 345, 346
блок команд, 49
блок пам'яті, 217
блок-схема, 9
булеві операції, 73

В

вектор, 149
визначення
– змінної, 51, 238, 346
– функції, 305, 339
властивості елементів керування, 30, 31, 33
– AutoSize, 27, 28, 30
– Color, 30, 34, 110
– Enabled, 28, 34
– Height, 228, 236
– Lines, 152, 430
– Text, 21, 30
– Width, 30, 33, 153, 160, 192, 450
вираз, 55
вирази, 47
вказівник, 206
вкладені цикли, 125
властивості алгоритмів, 9

Г

глобальні змінні, 308

Д

двовимірні масиви, 175
декремент, 56
дескриптор, 53, 288
динамічна пам'ять, 216
динамічний масив, 216, 218
динамічні бібліотеки, 350
директиви препроцесора, 341

Е

елементи керування, 25
ескейп-послідовність, 241

З

заголовні файли, 339
заголовок функції, 36, 193
зведення типів, 59
знаки операцій, 48, 55

змінна, 49

- автоматична, 312, 345
- глобальна, 308
- локальна, 308
- статична, 312, 338

I

ім'я (ідентифікатор) змінної, 49
іменована область, 347, 348
індекси масиву, 149
інкремент, 56

K

керувальні символи, 241
клас, 347
клас Console, 39
коментарі, 49
конкатенація, 289
консольний застосунок, 17
консольний режим, 17
константа, 54, 55
крок цикл, 12
купа, 216

L

лексема, 47
лексема, 254
лінійні алгоритми, 11, 12
літерал, 55
логічна операція, 73
логічне додавання, 73
логічне заперечення, 73
логічне множення, 73
логічні змінні, 73
логічні побітові операції, 75
локальні змінні, 308

M

масиви

- одновимірні, 149
- двовимірні, 175
- символьні, 247
- динамічні, 218, 229

математичні функції, 61, 465

матриці, 176

мітка, 72

мова програмування, 47

N

нульовий символ ("0"), 35, 55, 151, 241, 246, 255, 279

O

об'єднання, 363

області дії, 346

область видимості, 337

область видимості змінної, 345

оголошення змінної, 53

оголошення функції, 305, 339

одновимірний масив, 149

операнд, 11, 55

оператори, 55

- break, 95, 104, 131, 145, 146
- continue, 104, 146
- delete, 216, 218, 230
- do-while, 130
- for, 102
- goto, 72, 98
- if, 81
- namespace, 22, 39, 347, 348
- new, 216, 218, 229
- return, 255, 305, 306
- switch, 95
- using, 36
- using namespace, 19, 349, 450
- while, 130
- присвоювання, 58

операції

- . (крапка), 364, 462
- ->, 366, 462
- ::, 36, 347, 348
- sizeof, 53, 365, 444
- адресації &, 38, 207
- арифметичні, 56
- відношень, 72
- декремент --, 56, 210
- зсуву <<, >>, 76, 79
- інкремент ++, 56, 210
- логічні !, &&, ||, 73
- логічні побітові ~, &, |, ^, 75
- остача від ділення цілих чисел %, 55
- розадресація *, 207, 220, 314
- умовна (? :), 92
- явного зведення типів, 60

остача від ділення, 56

P

параметр циклу, 12

параметри функцій

- зі значеннями за промовчанням, 319
- змінної кількості, 332
- передавання за вказівником, 312
- передавання за посиланням, 312
- передавання за значенням, 312

перевантаження функцій, 332
перейменовування типів typedef, 362
перерахування, 376
перерахування enum, 52
перетин масиву, 177
події елементів керування, 26, 27

- Activated, 27
- Click, 27, 30
- DoubleClick, 27
- DragDrop, 27
- Enter, 27
- KeyDown, 27
- KeyPress, 27
- KeyUp, 27
- Load, 27, 153
- MouseDown, 27
- MouseMove, 27, 32
- MouseUp, 27
- TextChanged, 27

поля структури, 364
порожній оператор, 55
послідовності, 65
правила організації функцій, 307
препроцесор, 341
препроцесорні операції, 342
пріоритет операцій, 57, 80, 462
програма, 11
простори імен, 347
прототип функції, 305, 347

Р

рекурсивні функції, 322
розадресація, 207
розгалужені алгоритми, 11, 13
розгалуження, 71
розіменування вказівника, 207
розмір масиву, 150
рядки, 53, 151, 240, 246, 247, 248, 279
рядки string, 279
рядки String, 288
рядок символів, 247

С

символи ASCII, 240
сортування масиву, 168, 297
специфікатори формату, 38, 382, 390
специфікатори класів пам'яті, 345
статична змінна, 312, 338
статичні бібліотеки, 350
стек, 216, 308, 312, 323
структура, 364
схема алгоритму, 9

Т

таблиця ASCII-кодів, 240
текстові файли, 402
тіло циклу, 102
типи даних, 50

- bool, 52, 53, 72
- char, 50, 52, 241, 242
- double, 40, 50, 59, 213
- enum, 50, 52, 376
- float, 50, 54
- int, 50, 51, 59
- long, 50, 51, 59
- long double, 50, 51, 53
- short, 50, 51, 59
- signed, 51
- unsigned, 50, 51, 59
- void, 50, 52, 164
- wchar_t, 299

тривалість життя змінної, 345

Ф

файл, 347, 402, 403

- бінарні, 432
- текстові, 403

файлові потоки, 418
форма, 22, 29, 359
фрейм стека, 323
функції, 305, 347

- main, 36
- введення-виведення, 38, 249, 465
- відгуку на подію, 27, 32, 34, 306
- визначення, 305, 339, 357
- генерування випадкових чисел, 62, 230
- математичні, 61, 465
- перетворення числових типів, 40
- оголошення, 305, 339
- опрацювання дати і часу, 380
- опрацювання рядків char*, 250, 466
- опрацювання рядків string, 281, 468
- опрацювання символів, 243, 264
- опрацювання файлів, 403, 465
- перевантаження, 332
- передавання параметрів, 311
- правила організації, 305
- прототип, 305, 347
- рекурсивні, 322
- убудовані, 305, 307

Ц

цикл, 12, 13, 125
циклічні алгоритми, 12, 13

c