Міністерство освіти і науки України
Міжнародний гуманітарний університет
Кафедра комп’ютерних наук

Русу О.П.

АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ

Методичні вказівки до виконання практичних робіт
для здобувачів вищої освіти,
які навчаються за спеціальностями
галузі «Інформаційні технології»

Одеса 2025

Затверджено Вченою Радою Міжнародного гуманітарного університету.
Протокол № 5 від 20 січня 2025 р.

Русу О.П. Алгоритмізація та програмування. Методичні вказівки до виконання практичних робіт для здобувачів вищої освіти, які навчаються за спеціальностями галузі «Інформаційні технології» [Електронне видання]. Кафедра комп'ютерних наук Міжнародного гуманітарного університету. Одеса, 2025. 23 с.

Навчальна дисципліна «Алгоритмізація та програмування» є базовою складовою професійної підготовки фахівців зі спеціальності «Інженерія програмного забезпечення» та формує фундаментальні знання з алгоритмічного мислення і принципів розробки програмного забезпечення. У межах дисципліни вивчаються поняття алгоритму, його властивості та способи подання, базові алгоритмічні конструкції, типи даних, функції та масиви, а також принципи модульної побудови програм. Особлива увага приділяється формалізації задач, реалізації алгоритмів засобами мови програмування високого рівня та аналізу коректності й ефективності програмних рішень. Дисципліна закладає теоретичну та практичну основу для подальшого опанування фахових курсів і формує базові компетентності розробника програмного забезпечення.

ЗМІСТ

Практична робота 1. Побудова блок-схем алгоритмів розв'язання прикладних задач

Ключові положення

Алгоритм є формалізованим описом послідовності дій, виконання яких приводить до розв'язання певної задачі. У програмуванні алгоритм визначає логіку роботи майбутньої програми та структуру обробки даних. Перед реалізацією алгоритму мовою програмування доцільно виконати його попередній аналіз і представити у формалізованому вигляді. Одним із найбільш наочних способів представлення алгоритмів є використання блок-схем.

Блок-схема алгоритму є графічним зображенням послідовності виконання операцій. Вона складається зі стандартних графічних елементів, які з'єднуються між собою лініями переходів. Кожен елемент блок-схеми відповідає певному типу операції або етапу виконання алгоритму. Такий спосіб подання дозволяє наочно відобразити логіку роботи алгоритму, структуру розгалужень і повторень, а також взаємозв'язок між окремими етапами обчислення.

Використання блок-схем є важливим етапом алгоритмізації задач. Перед написанням програмного коду програміст повинен чітко визначити послідовність дій, які повинна виконувати програма. Побудова блок-схеми дозволяє перевірити правильність алгоритму, виявити можливі логічні помилки та оптимізувати структуру обчислень.

Для побудови блок-схем використовуються стандартні графічні символи. Найважливішими з них є елементи початку та завершення алгоритму, блоки виконання операцій, блоки введення і виведення даних, а також блоки перевірки умов.

Блок початку і завершення алгоритму використовується для позначення початку та завершення виконання програми. Цей елемент має форму овала або прямокутника із заокругленими кутами. У ньому зазвичай записується слово «Початок» або «Кінець».

Блок виконання операції використовується для позначення обчислень або інших дій над даними. Такий блок має форму прямокутника. У ньому можуть записуватися арифметичні вирази, присвоювання значень змінним або інші операції обробки даних.

Блок введення та виведення даних використовується для взаємодії алгоритму з користувачем або з іншими джерелами інформації. Він має форму паралелограма. У цьому блоці можуть записуватися операції введення значень змінних або виведення результатів обчислень.

Блок перевірки умови використовується для реалізації розгалужених алгоритмів. Він має форму ромба. Усередині такого блоку записується логічна умова, результат якої може бути істинним або хибним. Залежно від результату перевірки алгоритм переходить до відповідної гілки виконання.

Зв'язки між елементами блок-схеми здійснюються за допомогою стрілок. Стрілки показують напрямок виконання алгоритму і визначають послідовність переходів між блоками.

Під час побудови блок-схем необхідно дотримуватися певних правил. Алгоритм повинен мати один початковий і один завершальний блок. Напрямок виконання алгоритму зазвичай зображається зверху вниз або зліва направо. Усі блоки повинні бути з'єднані між собою таким чином, щоб було зрозуміло, у якому порядку виконуються операції.

Блок-схеми широко використовуються у процесі навчання програмуванню, оскільки вони дозволяють студентам краще зрозуміти структуру алгоритмів. Крім того, блок-схеми

застосовуються під час документування програмного забезпечення та під час проєктування складних програмних систем.

Побудова блок-схем є першим етапом переходу від опису задачі природною мовою до її реалізації у вигляді програмного коду. Після створення блок-схеми алгоритм може бути реалізований у вигляді програми мовою програмування.

Практичне завдання

1. Побудувати блок-схему алгоритму обчислення суми двох чисел, введених користувачем.
2. Побудувати блок-схему алгоритму обчислення значення математичного виразу

$$S = a + b * c$$

де значення змінних вводяться з клавіатури.

3. Побудувати блок-схему алгоритму визначення більшого з двох чисел.
4. Побудувати блок-схему алгоритму обчислення середнього арифметичного трьох чисел.
5. Побудувати блок-схему алгоритму обчислення площі прямокутника за заданими сторонами.

Методичні вказівки до виконання практичного завдання

Перед побудовою блок-схеми необхідно уважно проаналізувати умову задачі та визначити вхідні дані, які використовуються у програмі, а також результати, які повинні бути отримані. На цьому етапі доцільно сформулювати послідовність дій, які необхідно виконати для розв’язання задачі.

Після аналізу задачі формується алгоритм її розв’язання. Алгоритм повинен містити всі необхідні етапи обробки даних: введення початкових значень, виконання обчислень та виведення результатів.

Побудову блок-схеми слід починати з розміщення блоку початку алгоритму. Після цього додається блок введення даних, у якому зазначаються змінні, значення яких вводяться користувачем.

Наступним етапом є виконання обчислень. Для цього використовується блок виконання операції. У цьому блоці записується математичний вираз або інша операція, яка виконується над даними.

Якщо алгоритм містить перевірку умов, необхідно використати блок перевірки умови. У такому блоці записується логічний вираз. Від нього відходять дві гілки, що відповідають результатам перевірки умови.

Після виконання всіх необхідних обчислень алгоритм завершується блоком виведення результатів. У цьому блоці зазначається змінна або вираз, значення якого необхідно відобразити.

Останнім елементом блок-схеми є блок завершення алгоритму. Він позначає завершення виконання програми.

Під час побудови блок-схеми необхідно використовувати стрілки для позначення напрямку виконання алгоритму. Усі елементи повинні бути розташовані у логічній послідовності. Не допускається наявність блоків, які не мають входу або виходу.

Для побудови блок-схем можна використовувати графічні редактори, спеціалізовані програмні засоби або виконувати побудову вручну на папері. Важливо, щоб блок-схема була чіткою, зрозумілою та відповідала логіці алгоритму.

Контрольні питання

1. Що називається алгоритмом?
2. Які основні властивості алгоритмів?
3. Що таке блок-схема алгоритму?
4. Які основні графічні елементи використовуються у блок-схемах?
5. Яке призначення блоку введення і виведення даних?
6. Для чого використовується блок перевірки умови?
7. Які правила необхідно дотримуватися під час побудови блок-схем?
8. Яку роль відіграють блок-схеми у процесі розробки програмного забезпечення?

Зміст звіту

1. Назва та мета роботи.
2. Короткі теоретичні відомості про алгоритми та блок-схеми.
3. Блок-схеми алгоритмів розв’язання поставлених задач.
4. Пояснення логіки побудови кожного алгоритму.
5. Висновки за результатами виконання роботи.

Практична робота 2. Розробка алгоритму відповідно до етапів життєвого циклу програмного забезпечення

Ключові положення

Розробка програмного забезпечення є складним процесом, що включає низку взаємопов’язаних етапів. Для організації цього процесу використовується поняття життєвого циклу програмного забезпечення. Життєвий цикл визначає послідовність етапів, які проходить програмний продукт від моменту формування ідеї до завершення його використання або модернізації.

У загальному випадку життєвий цикл програмного забезпечення включає такі етапи: аналіз задачі та формування вимог, проєктування алгоритмів і структури програми, реалізацію програмного коду, тестування програмного забезпечення та подальший супровід програмного продукту. Кожен із цих етапів відіграє важливу роль у створенні надійної та ефективної програмної системи.

Першим етапом розробки є аналіз задачі. На цьому етапі визначаються вимоги до програмного забезпечення, уточнюється мета програми, визначаються вхідні дані та результати, які повинні бути отримані. Важливо чітко сформулювати умову задачі та зрозуміти, які саме операції повинна виконувати програма.

Наступним кроком є формалізація задачі. Формалізація полягає у переході від словесного опису задачі до її формального представлення. У процесі формалізації визначаються змінні, які використовуються у програмі, встановлюються залежності між ними та формуються математичні або логічні моделі задачі.

Формалізація задачі є важливим етапом алгоритмізації. Вона дозволяє точно визначити структуру обчислень та уникнути неоднозначності у трактуванні задачі. У результаті формалізації програміст отримує чітке уявлення про те, які дані необхідно обробляти та які результати повинна видавати програма.

Після формалізації задачі виконується розробка алгоритму. Алгоритм визначає послідовність дій, які необхідно виконати для отримання результату. На цьому етапі програміст визначає структуру обчислень, використання умовних операторів і циклів, а також порядок виконання операцій.

Алгоритм може бути поданий у вигляді словесного опису, псевдокоду або блок-схеми. Використання блок-схем дозволяє наочно представити логіку алгоритму і полегшує перевірку правильності його побудови.

Після розробки алгоритму виконується реалізація програмного коду. Реалізація передбачає запис алгоритму мовою програмування. У межах дисципліни алгоритмізації та програмування для реалізації алгоритмів зазвичай використовується мова програмування C або C++.

Під час написання програмного коду використовуються змінні, оператори присвоювання, арифметичні операції, умовні оператори та оператори циклів. Важливо, щоб структура програми відповідала логіці алгоритму, розробленого на попередньому етапі.

Після завершення реалізації програми виконується її тестування. Метою тестування є перевірка правильності роботи програми та виявлення можливих помилок. Під час тестування використовуються різні набори вхідних даних, що дозволяє перевірити роботу програми у різних ситуаціях.

Останнім етапом життєвого циклу програмного забезпечення є супровід програмного продукту. Супровід включає виправлення помилок, удосконалення функціональності програми та її адаптацію до нових умов використання.

Таким чином, формалізація задачі та розробка алгоритму є важливими етапами створення програмного забезпечення. Саме на цих етапах формується логічна структура майбутньої програми, визначаються необхідні дані та способи їх обробки.

Практичне завдання

1. Проаналізувати прикладну задачу та виконати її формалізацію.
2. Визначити вхідні та вихідні дані задачі.
3. Побудувати математичну або логічну модель задачі.
4. Розробити алгоритм розв’язання задачі у вигляді блок-схеми.
5. Записати алгоритм у вигляді псевдокоду.

Приклад задачі для виконання роботи: Розробити алгоритм обчислення значення функції

$$y = (a + b) / c.$$

де значення змінних (a), (b) та (c) вводяться користувачем.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно уважно проаналізувати умову задачі. Потрібно визначити, які величини є вхідними даними, а які – результатами обчислення. На цьому етапі доцільно записати словесний опис задачі та визначити її мету.

Далі виконується формалізація задачі. Необхідно ввести позначення змінних, визначити їх типи та встановити математичні залежності між ними. Якщо задача містить математичну формулу, її потрібно записати у зручному для обчислення вигляді.

Після формалізації задачі розробляється алгоритм її розв'язання. Алгоритм повинен містити такі основні етапи: початок алгоритму, введення вхідних даних, виконання необхідних обчислень, виведення результатів та завершення алгоритму.

Для наочного представлення алгоритму необхідно побудувати блок-схему. У блок-схемі повинні бути використані стандартні графічні елементи, що відповідають різним типам операцій. Важливо забезпечити правильну послідовність переходів між блоками.

Після побудови блок-схеми алгоритм необхідно записати у вигляді псевдокоду. Псевдокод дозволяє представити алгоритм у текстовій формі, близькій до програмного коду, але без прив'язки до конкретної мови програмування.

Під час виконання роботи слід звернути увагу на логічну послідовність виконання алгоритму. Усі операції повинні бути чітко визначені, а алгоритм повинен завершуватися отриманням результату.

Контрольні питання

1. Що розуміють під життєвим циклом програмного забезпечення?
2. Які основні етапи життєвого циклу програмного забезпечення?
3. Що таке формалізація задачі?
4. Які етапи включає процес алгоритмізації?
5. Які способи подання алгоритмів використовуються у програмуванні?
6. Яку роль відіграє блок-схема під час розробки алгоритму?
7. Чим відрізняється псевдокод від програмного коду?
8. Чому важливо виконувати аналіз задачі перед написанням програми?

Зміст звіту

1. Назва та мета роботи.
2. Формулювання прикладної задачі.
3. Формалізація задачі (визначення змінних і математичної моделі).
4. Блок-схема алгоритму розв'язання задачі.
5. Псевдокод алгоритму.
6. Висновки за результатами виконання роботи.

Практична робота 3. Реалізація лінійних алгоритмів та введення-виведення даних

Ключові положення

Після розробки алгоритму наступним етапом створення програмного забезпечення є його реалізація мовою програмування. У межах дисципліни алгоритмізації та програмування для реалізації алгоритмів використовується мова програмування С або С++, яка є однією з найпоширеніших мов системного та прикладного програмування. Ця мова характеризується високою ефективністю виконання програм і широкими можливостями роботи з пам'яттю та даними.

Найпростішими алгоритмами є лінійні алгоритми. Лінійний алгоритм – це алгоритм, у якому всі команди виконуються послідовно одна за одною без розгалужень та повторень. У таких алгоритмах порядок виконання операцій визначений однозначно, і кожна інструкція виконується лише один раз.

Лінійні алгоритми широко використовуються для розв'язання задач, пов'язаних із обчисленням математичних виразів, перетворенням даних або виконанням простих обчислювальних операцій. У програмному коді лінійні алгоритми реалізуються у вигляді послідовності операторів.

Для реалізації програм використовується інтегроване середовище розробки Visual Studio Code. Це сучасний редактор програмного коду, який підтримує велику кількість мов програмування, зокрема С і С++. Visual Studio Code забезпечує зручні інструменти для написання, компіляції та налагодження програм.

Перед початком роботи необхідно встановити компілятор С/С++. Для цього у системі Windows зазвичай використовується пакет MinGW або інший сумісний компілятор. Після встановлення компілятора Visual Studio Code може використовувати його для компіляції програм.

Програма мовою С або С++ має певну структуру. Виконання програми починається з функції main. Саме у цій функції розміщується основна частина алгоритму. У програмі можуть використовуватися змінні різних типів, арифметичні операції, оператори введення і виведення даних.

Важливим елементом програмування є робота з даними. Дані, які використовуються у програмі, зберігаються у змінних. Кожна змінна має певний тип, який визначає, які значення вона може зберігати. До основних типів даних належать цілі числа, дійсні числа та символи.

Для взаємодії програми з користувачем використовуються операції введення та виведення даних. Введення даних дозволяє користувачу передати програмі необхідні значення, які будуть використані під час обчислень. Виведення даних використовується для відображення результатів виконання програми.

У мовах С і С++ введення і виведення даних може здійснюватися різними способами. У мові С для цього використовуються стандартні функції введення і виведення. У мові С++ широко застосовуються потоки введення і виведення.

Під час написання програм важливо правильно організувати процес введення даних. Програма повинна чітко повідомляти користувача про те, які значення необхідно ввести. Це дозволяє уникнути помилок під час виконання програми.

Лінійні алгоритми зазвичай складаються з трьох основних етапів: введення початкових даних, виконання обчислень та виведення результатів. Така структура характерна для більшості простих програм.

Наприклад, програма може обчислювати значення математичного виразу. Спочатку програма запитує у користувача значення змінних, потім виконує необхідні обчислення і після цього відображає результат.

Під час реалізації лінійних алгоритмів необхідно звертати увагу на порядок виконання арифметичних операцій. Пріоритет операцій визначає, у якій послідовності виконуються обчислення у математичних виразах.

Окрім арифметичних операцій, у програмі використовуються оператори присвоювання. Оператор присвоювання використовується для збереження результату обчислення у змінній.

Під час створення програм у середовищі Visual Studio Code важливо правильно організувати процес компіляції. Спочатку програмний код зберігається у текстовому файлі з відповідним розширенням. Після цього компілятор перетворює програмний код у виконуваний файл.

У разі наявності помилок у програмному коді компілятор повідомляє про них. Такі помилки називаються синтаксичними. Після виправлення помилок програма може бути успішно скомпільована і виконана.

Таким чином, реалізація лінійних алгоритмів є важливим етапом формування практичних навичок програмування. Вона дозволяє студентам засвоїти основні принципи роботи з програмним кодом, змінними та операціями введення і виведення даних.

Практичне завдання

1. Створити програму для обчислення суми двох чисел, введених користувачем.
2. Створити програму для обчислення значення математичного виразу

$$y = (a + b) / c$$

де значення змінних вводяться користувачем.

3. Розробити програму для обчислення середнього арифметичного трьох чисел.
4. Розробити програму для обчислення площі прямокутника за заданими сторонами.
5. Розробити програму для обчислення значення функції

$$S = a^2 + b^2$$

де значення змінних вводяться з клавіатури.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно підготувати середовище розробки Visual Studio Code. Після запуску середовища слід створити новий файл програмного коду з розширенням .c або .cpp.

У програмі необхідно оголосити змінні, які будуть використовуватися для зберігання вхідних даних та результатів обчислень. Тип змінних повинен відповідати характеру даних, які вони зберігають.

Далі у програмі необхідно реалізувати введення даних з клавіатури. Після введення значень змінних виконується обчислення відповідного математичного виразу.

Результат обчислення необхідно зберегти у змінній і після цього вивести на екран. Для цього використовуються стандартні засоби виведення даних.

Після написання програми необхідно зберегти файл і виконати компіляцію за допомогою компілятора C/C++. Якщо у програмі відсутні синтаксичні помилки, компілятор створить виконуваний файл.

Після компіляції програму необхідно запустити і перевірити її роботу на кількох наборах вхідних даних. Отримані результати повинні відповідати математичним розрахункам.

У разі виникнення помилок необхідно перевірити програмний код, виправити помилки та повторно виконати компіляцію.

Контрольні питання

1. Що називається лінійним алгоритмом?
2. Які основні етапи реалізації лінійного алгоритму?
3. Яку структуру має програма мовою C або C++?
4. Яку роль відіграє функція main?
5. Для чого використовуються змінні у програмі?
6. Які операції використовуються для введення і виведення даних?
7. Які етапи включає процес компіляції програми?
8. Які помилки можуть виникати під час компіляції програм?

Зміст звіту

1. Назва та мета роботи.
2. Короткі теоретичні відомості про лінійні алгоритми.
3. Програмний код розроблених програм.
4. Результати виконання програм з різними наборами вхідних даних.
5. Висновки за результатами виконання роботи.

Практична робота 4. Використання арифметичних операцій і перетворення типів даних у програмах

Ключові положення

Під час розробки програмного забезпечення важливу роль відіграє виконання арифметичних обчислень. Більшість прикладних програм пов’язана з обробкою числових даних, тому програміст повинен добре розуміти правила використання арифметичних операцій, особливості роботи з різними типами даних та механізми їх перетворення.

У мовах програмування C і C++ передбачено стандартний набір арифметичних операцій, які дозволяють виконувати основні математичні обчислення. До таких операцій належать додавання, віднімання, множення, ділення та знаходження остачі від ділення. Ці операції використовуються для обробки числових даних у більшості програм.

Операція додавання використовується для обчислення суми двох значень. Операція віднімання дозволяє визначити різницю між значеннями. Операція множення застосовується для обчислення добутку чисел. Операція ділення дозволяє знайти результат ділення одного числа на інше. Операція знаходження остачі використовується лише для цілих чисел і дозволяє визначити залишок від ділення.

Важливою особливістю виконання арифметичних операцій є пріоритет операцій. Пріоритет визначає порядок виконання обчислень у математичних виразах. Наприклад, операції множення і ділення виконуються раніше, ніж операції додавання і віднімання. Для зміни порядку виконання обчислень використовуються круглі дужки.

У програмуванні часто використовуються складні арифметичні вирази, які містять кілька операцій. У таких випадках програміст повинен уважно контролювати порядок виконання обчислень, щоб отримати правильний результат.

Ще одним важливим аспектом роботи з арифметичними операціями є використання різних типів даних. У мовах C і C++ передбачено кілька числових типів даних. До найпоширеніших належать цілі числа та дійсні числа. Цілі числа використовуються для представлення значень без дробової частини, тоді як дійсні числа дозволяють працювати з дробовими значеннями.

Під час виконання обчислень важливо враховувати типи змінних, що беруть участь у виразі. Якщо обидва операнди є цілими числами, результат обчислення також буде цілим числом. У такому випадку дробова частина результату буде відкинута.

Наприклад, якщо виконати ділення двох цілих чисел, результатом буде ціле число, навіть якщо математично результат має дробову частину. Це є важливою особливістю роботи арифметичних операцій у програмуванні.

Для отримання точного результату обчислення необхідно використовувати змінні дійсного типу. У цьому випадку результат обчислення буде містити дробову частину.

У процесі обчислень часто виникає необхідність перетворення одного типу даних на інший. Такий процес називається перетворенням типів даних. Перетворення типів дозволяє змінювати спосіб представлення даних у пам’яті комп’ютера.

Перетворення типів може виконуватися автоматично або явно. Автоматичне перетворення типів відбувається під час виконання арифметичних операцій, якщо у виразі використовуються змінні різних типів. У цьому випадку компілятор автоматично приводить значення до відповідного типу.

Наприклад, якщо у виразі беруть участь ціле число і дійсне число, результат обчислення буде дійсним числом. Це пов’язано з тим, що дійсний тип має вищу точність представлення даних.

Явне перетворення типів виконується програмістом. Воно використовується у тих випадках, коли необхідно спеціально змінити тип значення під час виконання програми. Явне перетворення типів дозволяє контролювати процес обчислення та уникати помилок, пов’язаних із неправильним типом даних.

Під час програмування важливо правильно використовувати перетворення типів, оскільки неправильне перетворення може призвести до втрати точності або до некоректних результатів обчислень.

Ще одним важливим аспектом є використання арифметичних операцій у поєднанні з операторами присвоювання. Оператор присвоювання дозволяє зберігати результат обчислення у змінній. Після виконання операції нове значення змінної може використовуватися у подальших обчисленнях.

У процесі розробки програм необхідно також враховувати можливість виникнення помилок під час обчислень. Наприклад, ділення на нуль є недопустимою операцією і може призвести до аварійного завершення програми. Саме тому програміст повинен перевіряти вхідні дані перед виконанням обчислень.

У середовищі Visual Studio Code написання та тестування програм здійснюється у текстових файлах з розширенням .c або .cpp. Після написання програмного коду програма компілюється за допомогою компілятора C/C++, після чого може бути виконана у командному рядку або у вбудованому терміналі середовища.

Таким чином, використання арифметичних операцій і правильна робота з типами даних є важливою складовою програмування. Розуміння правил виконання арифметичних операцій, пріоритету обчислень та механізмів перетворення типів дозволяє створювати коректні та ефективні програми.

Практичне завдання

1. Розробити програму для обчислення значення математичного виразу

$$S = (a + b) / c$$

де значення змінних вводяться користувачем.

2. Розробити програму для обчислення значення виразу

$$y = (a + b) / (a – b)$$

3. Створити програму для обчислення середнього арифметичного трьох чисел.

4. Розробити програму, яка визначає результат ділення двох чисел і виводить як результат ділення, так і остачу від ділення.

5. Створити програму, у якій використовується явне перетворення типів під час виконання арифметичних операцій.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно відкрити середовище Visual Studio Code і створити новий файл програмного коду з розширенням .c або .cpp.

У програмі необхідно оголосити змінні, які використовуються для зберігання вхідних даних і результатів обчислення. Тип змінних повинен відповідати характеру обчислень.

Після оголошення змінних необхідно реалізувати введення значень змінних з клавіатури. Після цього виконується обчислення відповідного арифметичного виразу.

Під час виконання обчислень необхідно враховувати порядок виконання операцій. У разі необхідності слід використовувати дужки для зміни порядку обчислень.

У завданнях, що передбачають використання різних типів даних, необхідно виконати перетворення типів. Це дозволить отримати коректний результат обчислення.

Після виконання обчислень результат необхідно вивести на екран. Для цього використовуються стандартні засоби виведення даних.

Після написання програми необхідно виконати її компіляцію. У разі виникнення помилок компілятор повідомить про їх наявність. Після виправлення помилок програму необхідно повторно скомпілювати і запустити.

Отримані результати необхідно перевірити на кількох наборах вхідних даних.

Контрольні питання

1. Які арифметичні операції використовуються у мовах C і C++?
2. Що визначає пріоритет арифметичних операцій?
3. Які типи даних використовуються для представлення чисел?
4. У чому полягає відмінність між цілими і дійсними числами?
5. Що таке автоматичне перетворення типів даних?
6. Що таке явне перетворення типів даних?
7. Які помилки можуть виникати під час виконання арифметичних операцій?
8. Чому важливо враховувати типи змінних під час виконання обчислень?

Зміст звіту

1. Назва та мета роботи.
2. Короткі теоретичні відомості про арифметичні операції та типи даних.
3. Програмний код розроблених програм.
4. Результати виконання програм.
5. Висновки за результатами виконання роботи.

Практична робота 5. Застосування логічних операцій та операцій відношення у програмуванні

Ключові положення

Під час розробки програмного забезпечення важливу роль відіграє аналіз умов та прийняття рішень залежно від значень змінних. У програмуванні це реалізується за допомогою логічних виразів, які формуються з використанням операцій відношення та логічних операцій. Такі операції дозволяють перевіряти виконання певних умов і використовуються під час реалізації розгалужених алгоритмів.

Операції відношення використовуються для порівняння значень змінних або результатів обчислення виразів. Результатом виконання операції відношення є логічне значення – істина або хиба. У мовах програмування C і C++ для позначення істинного значення використовується ненульове значення, а для хибного – нуль.

До основних операцій відношення належать перевірка рівності, перевірка нерівності, перевірка більшого або меншого значення, а також перевірка більшого або рівного і меншого або рівного значення. Ці операції дозволяють виконувати різноманітні перевірки під час виконання програми.

Операції відношення широко застосовуються у програмуванні для перевірки значень змінних перед виконанням певних дій. Наприклад, програма може перевіряти, чи перевищує певне значення заданий поріг, чи є число додатним або від’ємним, або чи належить значення до певного інтервалу.

У більш складних ситуаціях однієї операції відношення недостатньо. У таких випадках використовуються логічні операції, які дозволяють об'єднувати кілька умов у єдиний логічний вираз. Логічні операції використовуються для формування складних умов перевірки.

До основних логічних операцій належать логічне множення, логічне додавання та логічне заперечення. Логічне множення використовується для перевірки одночасного виконання кількох умов. Результат такого виразу буде істинним лише тоді, коли всі умови є істинними.

Логічне додавання використовується для перевірки виконання хоча б однієї з кількох умов. Якщо хоча б одна з умов є істинною, результат усього логічного виразу також буде істинним.

Логічне заперечення використовується для зміни логічного значення на протилежне. Якщо початкове значення є істинним, після застосування операції заперечення воно стає хибним, і навпаки.

Логічні операції дозволяють будувати складні логічні вирази, що складаються з кількох умов. Такі вирази широко використовуються під час реалізації умовних операторів.

Під час побудови логічних виразів необхідно враховувати пріоритет виконання логічних операцій. Спочатку виконуються операції заперечення, потім операції логічного множення, і лише після цього виконуються операції логічного додавання. Для зміни порядку виконання логічних операцій використовуються круглі дужки.

Правильне використання логічних операцій дозволяє створювати гнучкі алгоритми обробки даних. Наприклад, програма може перевіряти одночасно кілька умов перед виконанням певної операції або виконувати різні дії залежно від результатів перевірки.

Логічні вирази є основою реалізації розгалужених алгоритмів. У розгалужених алгоритмах подальший хід виконання програми залежить від результату перевірки певної умови. Саме тому програміст повинен уміти правильно формувати логічні вирази.

У програмуванні мовами C і C++ логічні вирази можуть використовуватися у різних конструкціях керування виконанням програми. Найчастіше вони застосовуються в умовних операторах і в операторах циклу.

Під час написання програм у середовищі Visual Studio Code програміст створює текстовий файл із програмним кодом, після чого виконує компіляцію програми за допомогою компілятора C/C++. У разі відсутності синтаксичних помилок компілятор створює виконуваний файл, який може бути запущений для перевірки роботи програми.

Під час тестування програм, що використовують логічні операції, необхідно перевіряти різні варіанти вхідних даних. Це дозволяє переконатися у правильності роботи логічних виразів та коректності виконання умов.

Таким чином, логічні операції та операції відношення є важливими інструментами програмування. Вони дозволяють аналізувати значення змінних, формувати складні умови перевірки та реалізовувати алгоритми, у яких порядок виконання дій залежить від певних умов.

Практичне завдання

1. Розробити програму, яка визначає, чи є введене число додатним, від'ємним або рівним нулю.

2. Створити програму для визначення більшого з двох чисел.

3. Розробити програму, яка перевіряє, чи належить число заданому інтервалу.

4. Створити програму для визначення, чи є введене число парним або непарним.

5. Розробити програму, яка перевіряє виконання складної умови, що містить кілька операцій відношення та логічних операцій.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно відкрити середовище розробки Visual Studio Code та створити новий файл програмного коду з розширенням .c або .cpp.

У програмі необхідно оголосити змінні, які використовуються для зберігання вхідних даних. Тип змінних повинен відповідати типу даних, що вводяться користувачем.

Після цього необхідно реалізувати введення значень змінних з клавіатури. Далі у програмі формуються логічні вирази, що використовують операції відношення та логічні операції.

Під час побудови логічних виразів необхідно уважно перевіряти правильність запису умов і враховувати порядок виконання логічних операцій. У разі необхідності слід використовувати дужки для уточнення порядку обчислень.

Після перевірки умов програма повинна вивести відповідне повідомлення про результат перевірки. Для цього використовуються стандартні засоби виведення даних.

Після написання програмного коду необхідно виконати компіляцію програми. У разі виникнення помилок необхідно виправити програмний код і повторно виконати компіляцію.

Після успішної компіляції програму необхідно протестувати на кількох наборах вхідних даних, щоб перевірити правильність роботи логічних виразів.

Контрольні питання

1. Що таке операції відношення?
2. Які операції відношення використовуються у мовах C і C++?
3. Що таке логічні операції?
4. Які основні логічні операції використовуються у програмуванні?
5. Який результат виконання логічного виразу?
6. Який порядок виконання логічних операцій?
7. Для чого використовуються логічні вирази у програмуванні?
8. Чому важливо тестувати програми з різними наборами вхідних даних?

Зміст звіту

1. Назва та мета роботи.
2. Короткі теоретичні відомості про логічні операції та операції відношення.
3. Програмний код розроблених програм.
4. Результати виконання програм.
5. Висновки за результатами виконання роботи.

Практична робота 6. Реалізація розгалужених алгоритмів із використанням умовних операторів

Ключові положення

Під час розробки програмного забезпечення часто виникають задачі, у яких подальший хід виконання алгоритму залежить від певних умов. У таких випадках використовуються розгалужені алгоритми. Розгалужений алгоритм – це алгоритм, у якому залежно від результату перевірки певної умови виконуються різні послідовності дій.

Реалізація розгалужених алгоритмів у мовах програмування здійснюється за допомогою умовних операторів. У мовах C і C++ для цього використовуються оператор if та оператор switch. Ці оператори дозволяють змінювати порядок виконання інструкцій програми залежно від значень змінних або результатів обчислення логічних виразів.

Найбільш поширеним є умовний оператор if. Він використовується для перевірки логічної умови. Якщо умова є істинною, виконується певний блок інструкцій. Якщо умова є хибною, виконання програми переходить до наступних інструкцій після умовного оператора.

Існує кілька форм використання оператора if. Найпростішою є скорочена форма умовного оператора, у якій передбачено виконання дії лише у випадку істинності умови. Якщо умова не виконується, програма просто переходить до наступних інструкцій.

Більш поширеною є повна форма умовного оператора. У цьому випадку передбачено виконання різних дій залежно від результату перевірки умови. Якщо умова є істинною, виконується один блок інструкцій, а якщо хибною – інший.

У програмуванні часто виникають задачі, що потребують перевірки кількох умов. Для реалізації таких алгоритмів використовуються вкладені умовні оператори. У вкладених конструкціях один умовний оператор розміщується всередині іншого. Це дозволяє реалізувати складні алгоритми прийняття рішень.

Вкладені умовні оператори використовуються, наприклад, під час визначення значення функції, яка залежить від кількох умов, або під час класифікації даних за різними критеріями. Однак під час використання вкладених конструкцій необхідно стежити за зрозумілістю програмного коду, оскільки велика кількість вкладених умов може ускладнити структуру програми.

Ще одним засобом реалізації розгалужених алгоритмів є оператор switch. Цей оператор використовується у випадках, коли необхідно обрати одну з кількох альтернатив залежно від значення змінної. Оператор switch зручний для реалізації алгоритмів, у яких значення змінної може набувати кількох фіксованих значень.

Під час використання оператора switch значення змінної порівнюється з кількома варіантами. Якщо значення змінної збігається з одним із варіантів, виконується відповідний блок інструкцій. Такий підхід дозволяє зробити програмний код більш структурованим та зрозумілим.

Оператори розгалуження широко використовуються у програмуванні. Вони застосовуються для перевірки правильності введених даних, реалізації математичних функцій, обробки результатів обчислень та виконання різних дій залежно від умов.

Під час написання програм у середовищі Visual Studio Code програміст створює файл програмного коду з розширенням .c або .cpp. Після написання коду програма компілюється за допомогою компілятора C/C++, після чого може бути запущена для перевірки її роботи.

Під час тестування програм, що містять умовні оператори, необхідно перевіряти всі можливі варіанти виконання алгоритму. Це дозволяє переконатися, що програма правильно реагує на різні значення вхідних даних.

Таким чином, умовні оператори є важливим інструментом реалізації розгалужених алгоритмів. Вони дозволяють змінювати порядок виконання інструкцій залежно від результатів перевірки логічних умов і забезпечують гнучкість програмного коду.

Практичне завдання

1. Розробити програму для визначення більшого з двох чисел.
2. Створити програму, яка визначає знак введеного числа (додатне, від'ємне або нуль).
3. Розробити програму для визначення, чи належить число заданому інтервалу.
4. Створити програму, яка обчислює значення функції залежно від значення змінної.
5. Розробити програму, у якій використовується оператор switch для вибору дії залежно від введеного значення.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно відкрити середовище розробки Visual Studio Code та створити новий файл програмного коду з розширенням .c або .cpp.

У програмі необхідно оголосити змінні, які використовуються для зберігання вхідних даних. Після цього необхідно реалізувати введення значень змінних з клавіатури.

Далі у програмі необхідно реалізувати перевірку умов за допомогою умовного оператора if або оператора switch. Логічні вирази повинні бути сформовані таким чином, щоб правильно відображати умову задачі.

Після виконання перевірки умов програма повинна виконати відповідні дії та вивести результат на екран. Для цього використовуються стандартні засоби виведення даних.

Після написання програмного коду необхідно виконати компіляцію програми. У разі виникнення помилок необхідно перевірити програмний код, виправити помилки та повторно виконати компіляцію.

Після успішної компіляції програму необхідно протестувати на кількох наборах вхідних даних. Особливу увагу слід приділити перевірці різних варіантів виконання умов.

Контрольні питання

1. Що називається розгалуженим алгоритмом?
2. Яке призначення умовного оператора if?
3. Які форми умовного оператора if використовуються у програмуванні?
4. Що таке вкладені умовні оператори?
5. У яких випадках використовується оператор switch?
6. Які переваги має оператор switch порівняно з оператором if?
7. Чому важливо перевіряти всі можливі варіанти виконання умов?
8. Яку роль відіграють логічні вирази у розгалужених алгоритмах?

Зміст звіту

1. Назва та мета роботи.
2. Короткі теоретичні відомості про розгалужені алгоритми.
3. Програмний код розроблених програм.
4. Результати виконання програм.
5. Висновки за результатами виконання роботи.

Практична робота 7. Програмування задач із використанням оператора вибору

Ключові положення

У процесі розробки програмного забезпечення часто виникають задачі, у яких необхідно виконати одну з кількох можливих дій залежно від значення певної змінної. У таких випадках використовуються алгоритми вибору. Алгоритм вибору передбачає виконання різних гілок алгоритму залежно від значення змінної або результату перевірки певної умови.

У мовах програмування C і C++ для реалізації алгоритмів вибору використовується оператор switch. Цей оператор дозволяє організувати вибір одного з кількох варіантів виконання програми залежно від значення змінної.

Оператор switch використовується у випадках, коли змінна може набувати кількох дискретних значень і для кожного з цих значень необхідно виконати певну послідовність дій. Такий підхід дозволяє значно спростити структуру програмного коду порівняно з використанням великої кількості умовних операторів.

Загальна структура оператора switch складається з виразу, значення якого перевіряється, а також кількох варіантів виконання, які відповідають можливим значенням цього виразу. Кожен варіант виконання позначається спеціальною міткою.

Після визначення значення виразу програма переходить до відповідного варіанту виконання і виконує інструкції, що відповідають цьому значенню. Якщо жоден із варіантів не відповідає значенню змінної, може бути використаний спеціальний блок, що визначає дії за замовчуванням.

Оператор switch є особливо зручним у задачах, пов’язаних із вибором дій залежно від введеного користувачем значення. Наприклад, за допомогою цього оператора можна реалізувати просте меню програми, де кожен пункт меню відповідає певній операції.

У мовах C і C++ значення, яке перевіряється оператором switch, повинно бути цілого або символьного типу. Кожен варіант виконання визначається константним значенням, з яким порівнюється значення змінної.

Під час реалізації оператора switch важливо правильно організувати завершення виконання кожного варіанту. Якщо цього не зробити, програма може продовжити виконання наступних варіантів, що призведе до некоректної роботи програми.

Застосування оператора switch дозволяє зробити програмний код більш структурованим і зрозумілим. Це особливо важливо у випадках, коли необхідно обробляти велику кількість можливих варіантів значень.

Під час розробки програм у середовищі Visual Studio Code програміст створює файл програмного коду з розширенням .c або .cpp. Після написання програми необхідно виконати її компіляцію за допомогою компілятора C/C++.

Після успішної компіляції програму необхідно запустити і перевірити її роботу на різних наборах вхідних даних. Особливу увагу слід приділити перевірці всіх можливих варіантів виконання оператора вибору.

Таким чином, оператор switch є важливим засобом реалізації алгоритмів вибору у програмуванні. Його використання дозволяє ефективно організувати виконання різних дій залежно від значення змінної та забезпечує зрозумілу структуру програмного коду.

Практичне завдання

1. Розробити програму, що реалізує просте меню вибору арифметичної операції. Користувач вводить номер операції, після чого програма виконує відповідну арифметичну дію над двома числами.
2. Створити програму, яка за номером дня тижня визначає його назву.
3. Розробити програму, що визначає пору року за номером місяця.
4. Створити програму, яка за введеним символом визначає тип операції (арифметична операція, символ порівняння або інший символ).
5. Розробити програму, яка за номером пункту меню виконує відповідну дію над введеними даними.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно відкрити середовище Visual Studio Code та створити новий файл програмного коду з розширенням .c або .cpp.

У програмі необхідно оголосити змінні, які використовуються для зберігання введених користувачем даних. Після цього необхідно реалізувати введення значення змінної, яка визначає варіант виконання програми.

Далі у програмі необхідно використати оператор switch для реалізації вибору відповідної дії. Для кожного можливого значення змінної необхідно передбачити окремий варіант виконання програми.

У кожному варіанті виконання повинні бути реалізовані відповідні обчислення або дії над даними. Після виконання обчислень необхідно вивести результат на екран.

Для випадків, коли введене значення не відповідає жодному з передбачених варіантів, необхідно передбачити обробку помилки введення.

Після написання програмного коду необхідно виконати компіляцію програми. У разі виникнення помилок необхідно виправити програмний код та повторно виконати компіляцію.

Після успішної компіляції програму необхідно протестувати на різних наборах вхідних даних, щоб перевірити правильність роботи оператора вибору.

Контрольні питання

1. Що називається алгоритмом вибору?
2. Для чого використовується оператор switch?
3. Які типи даних можуть використовуватися у виразі оператора switch?
4. Яке призначення варіантів виконання у операторі switch?
5. Для чого використовується блок виконання за замовчуванням?

6. У чому полягають переваги оператора switch порівняно з використанням кількох умовних операторів?

7. Які помилки можуть виникати під час реалізації оператора вибору?

8. Чому важливо тестувати всі варіанти виконання оператора switch?

Зміст звіту

1. Назва та мета роботи.
2. Короткі теоретичні відомості про оператор вибору.
3. Програмний код розроблених програм.
4. Результати виконання програм з різними наборами вхідних даних.
5. Висновки за результатами виконання роботи.

Практична робота 8. Розробка програм з використанням циклу з параметром та обчислення сум і добутків

Ключові положення

Під час розв'язання багатьох прикладних задач у програмуванні виникає необхідність багаторазового виконання однакових або подібних операцій. Для реалізації таких алгоритмів використовуються циклічні конструкції. Цикл – це конструкція керування виконанням програми, яка забезпечує повторення певної групи інструкцій задану кількість разів або до виконання певної умови.

У мовах програмування C і C++ існує кілька типів циклів. Одним із найбільш поширених є цикл з параметром, який реалізується за допомогою оператора for. Цей тип циклу використовується у випадках, коли кількість повторень заздалегідь відома або може бути визначена під час початку виконання алгоритму.

Оператор for дозволяє компактно записати всі елементи керування циклом у одному рядку. У загальному вигляді структура цього оператора містить три основні частини: ініціалізацію змінної циклу, перевірку умови виконання циклу та зміну значення змінної циклу після кожної ітерації.

Змінна циклу використовується для керування кількістю повторень. Вона змінюється після кожної ітерації циклу і дозволяє програмі визначити, коли необхідно завершити виконання повторюваних дій.

Цикл з параметром широко використовується під час обробки послідовностей даних, обчислення математичних виразів, формування таблиць значень функцій та виконання інших задач, у яких необхідно виконати певну операцію багаторазово.

Однією з найпоширеніших задач, що реалізуються за допомогою циклів, є обчислення сум та добутків послідовностей чисел. Наприклад, програма може обчислювати суму перших n натуральних чисел або добуток елементів певної числової послідовності.

Під час обчислення сум використовується спеціальна змінна, яка називається накопичувачем суми. На початку виконання програми ця змінна отримує початкове значення, після чого на кожній ітерації циклу до неї додається нове значення.

Аналогічно під час обчислення добутків використовується змінна накопичувача добутку. На початку виконання програми вона отримує значення, що відповідає

нейтральному елементу операції множення. Під час кожної ітерації циклу поточне значення множиться на новий елемент послідовності.

Використання циклів дозволяє значно спростити реалізацію алгоритмів обробки числових послідовностей. Без використання циклів програмісту довелося б записувати однакові операції багаторазово, що значно ускладнило б структуру програмного коду.

Під час програмування важливо правильно визначити початкове значення змінної циклу, умову виконання циклу та правило зміни змінної після кожної ітерації. Помилки у визначенні цих параметрів можуть призвести до некоректної роботи програми або до нескінченного циклу.

Під час розробки програм у середовищі Visual Studio Code програміст створює файл програмного коду з розширенням .c або .cpp. Після написання програми необхідно виконати її компіляцію за допомогою компілятора C/C++.

Після успішної компіляції програму необхідно запустити і перевірити правильність її роботи. Особливу увагу слід приділити перевірці правильності обчислення сум і добутків для різних значень параметрів.

Таким чином, цикл з параметром є важливим інструментом програмування, що дозволяє ефективно реалізовувати алгоритми, які потребують багаторазового виконання одних і тих самих операцій. Його використання значно спрощує структуру програмного коду і дозволяє реалізовувати складні обчислювальні алгоритми.

Практичне завдання

1. Розробити програму для обчислення суми перших n натуральних чисел.
2. Створити програму для обчислення добутку перших n натуральних чисел.
3. Розробити програму для обчислення суми квадратів перших n натуральних чисел.
4. Створити програму для обчислення значення арифметичної прогресії з використанням циклу.
5. Розробити програму, яка обчислює суму значень функції для заданого діапазону значень аргументу.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно відкрити середовище Visual Studio Code та створити новий файл програмного коду з розширенням .c або .cpp.

У програмі необхідно оголосити змінні, які використовуються для зберігання значень параметрів, а також змінні для накопичення результатів обчислення.

Після цього необхідно реалізувати введення значення параметра, який визначає кількість повторень циклу. Далі у програмі використовується оператор for для реалізації циклічного виконання обчислень.

У тілі циклу необхідно виконати обчислення відповідного елемента послідовності та додати або помножити його на накопичувач результату.

Після завершення виконання циклу необхідно вивести отриманий результат на екран.

Після написання програмного коду необхідно виконати компіляцію програми. У разі виникнення помилок необхідно перевірити програмний код, виправити помилки та повторно виконати компіляцію.

Після успішної компіляції програму необхідно протестувати на кількох значеннях параметра. Отримані результати повинні відповідати математичним обчисленням.

Контрольні питання

1. Що називається циклом у програмуванні?
2. Для чого використовується оператор for?
3. Які основні елементи містить структура оператора for?
4. Що таке змінна циклу?
5. Що називається накопичувачем суми?
6. Що називається накопичувачем добутку?
7. Які помилки можуть виникати під час використання циклів?
8. Чому важливо правильно визначати умову завершення циклу?

Зміст звіту

1. Назва та мета роботи.
2. Короткі теоретичні відомості про цикли з параметром.
3. Програмний код розроблених програм.
4. Результати виконання програм з різними значеннями параметра.
5. Висновки за результатами виконання роботи.

Практична робота 9. Реалізація задач із використанням циклів while, do while та вкладених циклів

Ключові положення

Під час розробки програмного забезпечення часто виникає необхідність багаторазового виконання певних операцій. Для реалізації таких алгоритмів використовуються циклічні конструкції. Цикли дозволяють організувати повторення групи інструкцій до виконання певної умови або протягом заданої кількості ітерацій.

У мовах програмування C і C++ існує кілька типів циклів. До найпоширеніших належать цикли for, while та do while. Цикл for зазвичай використовується у випадках, коли кількість повторень відома заздалегідь. У випадках, коли кількість повторень визначається під час виконання програми, використовуються цикли while або do while.

Цикл while належить до циклів із передумовою. Це означає, що перед кожною ітерацією циклу виконується перевірка логічної умови. Якщо умова є істинною, виконується тіло циклу. Якщо ж умова є хибною, виконання циклу завершується.

Цикл while зручно використовувати у випадках, коли кількість повторень заздалегідь невідома. Наприклад, цикл може виконуватися до тих пір, поки користувач вводить певні значення або поки результат обчислення не задовольнить задану умову.

Іншим різновидом циклу є цикл do while. На відміну від циклу while, у цьому випадку перевірка умови виконується після виконання тіла циклу. Це означає, що тіло циклу буде виконано принаймні один раз незалежно від значення умови.

Цикл do while зручно використовувати у ситуаціях, коли певна операція повинна бути виконана хоча б один раз. Наприклад, такий цикл часто використовується для організації введення даних користувачем із перевіркою правильності введених значень.

Ще одним важливим поняттям є вкладені цикли. Вкладеним називається цикл, який розташований всередині іншого циклу. У цьому випадку під час кожної ітерації зовнішнього циклу виконується повний цикл виконання внутрішнього циклу.

Вкладені цикли широко використовуються у задачах обробки двовимірних структур даних, формування таблиць значень, обчислення сум або добутків елементів матриць, а також під час реалізації різних алгоритмів обробки послідовностей даних.

Під час використання вкладених циклів важливо правильно організувати змінні керування циклами. Для кожного циклу використовується окрема змінна, що визначає номер поточної ітерації.

Циклічні конструкції часто використовуються для формування числових таблиць, обробки наборів даних, реалізації математичних алгоритмів та створення різних структур виведення інформації.

Під час програмування необхідно уважно контролювати умови завершення циклів. Якщо умова завершення сформульована неправильно, програма може перейти у нескінченний цикл, що призведе до її некоректної роботи.

Під час розробки програм у середовищі Visual Studio Code програміст створює файл програмного коду з розширенням .c або .cpp. Після написання програми необхідно виконати її компіляцію за допомогою компілятора C/C++.

Після успішної компіляції програму необхідно протестувати на різних наборах вхідних даних. Особливу увагу слід приділити перевірці правильності роботи умов завершення циклів.

Таким чином, цикли while та do while є важливими інструментами реалізації алгоритмів із невизначеною кількістю повторень. Використання вкладених циклів дозволяє реалізовувати більш складні алгоритми обробки даних і створювати структури обчислень, що складаються з кількох рівнів повторення.

Практичне завдання

1. Розробити програму для обчислення суми натуральних чисел, введених користувачем. Введення чисел завершується введенням нуля.
2. Створити програму, яка обчислює суму чисел від 1 до n із використанням циклу while.
3. Розробити програму, яка виконує повторне введення числа до тих пір, поки користувач не введе додатне значення. Для реалізації використати цикл do while.
4. Створити програму, яка формує таблицю множення з використанням вкладених циклів.
5. Розробити програму, яка виводить прямокутну таблицю чисел із використанням вкладених циклів.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно відкрити середовище розробки Visual Studio Code та створити новий файл програмного коду з розширенням .c або .cpp.

У програмі необхідно оголосити змінні, які використовуються для зберігання значень параметрів, а також змінні для накопичення результатів обчислення.

Після цього необхідно реалізувати введення необхідних даних із клавіатури. Далі необхідно використати відповідний оператор циклу для організації повторюваних обчислень.

У завданнях, де кількість повторень залежить від умови, доцільно використовувати цикл while або do while. У завданнях, що передбачають виконання кількох циклів одночасно, необхідно використовувати вкладені цикли.

Під час реалізації вкладених циклів слід уважно контролювати змінні керування кожного циклу. Змінні різних циклів не повинні взаємно впливати одна на одну.

Після виконання циклічних обчислень результат необхідно вивести на екран. Для цього використовуються стандартні засоби виведення даних.

Після написання програмного коду необхідно виконати компіляцію програми. У разі виникнення помилок необхідно перевірити програмний код, виправити помилки та повторно виконати компіляцію.

Після успішної компіляції програму необхідно протестувати на різних наборах вхідних даних, щоб перевірити правильність роботи циклічних конструкцій.

Контрольні питання

1. Що називається циклом у програмуванні?
2. Які типи циклів використовуються у мовах C і C++?
3. Чим відрізняється цикл while від циклу do while?
4. Що таке цикл із передумовою?
5. Що таке цикл із післяумовою?
6. Що називається вкладеним циклом?
7. У яких задачах використовуються вкладені цикли?
8. Які помилки можуть виникати під час використання циклів?

Зміст звіту

1. Назва та мета роботи.
2. Короткі теоретичні відомості про цикли while, do while та вкладені цикли.
3. Програмний код розроблених програм.
4. Результати виконання програм.
5. Висновки за результатами виконання роботи.

Практична робота 10. Розробка програм із використанням функцій та передавання параметрів

Ключові положення

У процесі розробки програмного забезпечення важливого значення набуває структурованість програмного коду. Якщо програма складається з великої кількості інструкцій, записаних у межах однієї функції, її важко аналізувати, тестувати та модифікувати. Для подолання цієї проблеми у програмуванні використовуються функції. Функція є окремим іменованим блоком програмного коду, який виконує певну завершену

задачу. Використання функцій дає змогу розділити складну програму на менші логічні частини, кожна з яких відповідає за окремий етап обробки даних.

У мовах C і C++ будь-яка програма починає виконання з функції main. Саме вона є головною функцією програми. Однак для реалізації складніших алгоритмів доцільно створювати додаткові функції, які викликаються з main або з інших функцій. Такий підхід є основою модульної побудови програм.

Функція має ім’я, тип значення, яке повертається, а також список параметрів. Якщо функція повинна передавати результат своєї роботи у викликаючу частину програми, у її заголовку зазначається відповідний тип результату. Якщо ж функція виконує лише певні дії і не повертає значення, використовується тип void.

Під час роботи з функціями розрізняють прототип функції, її опис і виклик. Прототип функції є попереднім оголошенням, яке повідомляє компілятору ім’я функції, тип результату та список параметрів. Прототип розміщується до функції main або в окремому заголовковому файлі. Це дає змогу викликати функцію раніше, ніж у програмі буде наведено її повний опис.

Опис функції містить її заголовок і тіло. Заголовок задає тип результату, ім’я та список параметрів. У тілі функції записуються інструкції, які виконуються під час її виклику. Це можуть бути обчислення, перевірки умов, циклічні операції, робота з масивами та інші дії.

Виклик функції є передаванням керування від однієї частини програми до іншої. Під час виклику функції можуть передаватися значення параметрів, а після завершення її роботи програма повертається до точки виклику. Якщо функція повертає результат, цей результат може бути використаний у виразах, операторах присвоювання або переданий іншій функції.

Важливим поняттям під час використання функцій є параметри. Параметри забезпечують обмін даними між викликаючою частиною програми та функцією. У програмуванні розрізняють формальні та фактичні параметри. Формальні параметри вказуються у заголовку функції. Вони є локальними змінними цієї функції та використовуються для роботи з переданими значеннями. Фактичні параметри – це конкретні змінні, константи або вирази, які підставляються під час виклику функції.

У мовах C і C++ параметри можуть передаватися за значенням або за адресою. Передавання за значенням означає, що у функцію передається копія значення фактичного параметра. У цьому випадку зміна формального параметра всередині функції не впливає на значення змінної у викликаючій програмі. Такий спосіб є зручним у випадках, коли функція повинна лише використовувати отримані дані для обчислень.

Передавання за адресою означає, що у функцію передається адреса змінної у пам’яті. У цьому випадку функція отримує доступ до тієї самої змінної, що існує у викликаючій частині програми. Якщо значення змінної змінюється всередині функції, ці зміни будуть збережені і після завершення її виконання. Такий спосіб передавання параметрів доцільно використовувати тоді, коли функція повинна змінити значення змінних або повернути кілька результатів одночасно.

Під час програмування необхідно чітко розуміти, який спосіб передавання параметрів використовується в конкретній задачі. Якщо функція повинна лише обчислити результат на основі заданих значень, достатньо передавання за значенням. Якщо ж функція має змінити вхідні дані, слід використовувати передавання за адресою.

Використання функцій дає низку переваг. По-перше, зменшується дублювання програмного коду, оскільки одну й ту саму функцію можна викликати багаторазово. По-

друге, програма стає більш зрозумілою, оскільки кожна функція реалізує окрему підзадачу. По-третє, спрощується тестування, оскільки правильність роботи можна перевіряти не для всієї програми одразу, а для окремих функцій.

У середовищі Visual Studio Code програми мовою C/C++ створюються у текстових файлах з розширенням .c або .cpp. Після написання програмного коду файл компілюється за допомогою встановленого компілятора, наприклад GCC або G++. Під час компіляції перевіряється правильність запису функцій, відповідність типів параметрів і правильність викликів.

Таким чином, функції є важливим інструментом побудови структурованих програм. Вони дають змогу розділити задачу на підзадачі, організувати повторне використання коду та забезпечити зручний обмін даними між різними частинами програми.

Практичне завдання

1. Розробити програму, у якій функція обчислює суму двох чисел і повертає результат у головну програму.
2. Створити програму, у якій окрема функція обчислює площу прямокутника за двома сторонами.
3. Розробити програму, у якій функція визначає більше з двох чисел.
4. Створити програму, у якій функція змінює значення двох змінних за допомогою передавання параметрів за адресою.
5. Розробити програму, у якій одна функція обчислює середнє арифметичне трьох чисел, а інша – виводить результат на екран.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно відкрити середовище Visual Studio Code та створити новий файл програмного коду з розширенням .c або .cpp. Далі слід підключити необхідні бібліотеки та оголосити прототипи функцій, які будуть використовуватися у програмі.

Після цього необхідно реалізувати функцію main, у якій виконується введення вихідних даних, виклик допоміжних функцій та виведення результатів. Основні обчислення доцільно винести в окремі функції, щоб структура програми була більш зрозумілою та логічною.

Під час розробки функцій необхідно чітко визначити, які параметри повинні передаватися у функцію, який тип результату вона повертає, а також чи потрібна зміна значень змінних у викликаючій програмі. У випадках, коли функція лише обчислює результат, слід використовувати передавання параметрів за значенням. Якщо функція повинна змінити значення змінних, необхідно використовувати передавання за адресою.

Після написання функцій необхідно перевірити правильність їх виклику з функції main. Кількість і типи фактичних параметрів повинні відповідати формальним параметрам, зазначеним у прототипі та описі функції.

Особливу увагу слід приділити використанню оператора return. Якщо функція повертає результат, він повинен мати тип, що відповідає заголовку функції. Якщо функція не повертає значення, використовується тип void.

Після завершення написання програми необхідно зберегти файл та виконати компіляцію. У разі появи синтаксичних помилок слід перевірити правильність запису прототипів, заголовків функцій, списків параметрів та викликів функцій.

Після успішної компіляції програму необхідно протестувати на кількох наборах вхідних даних. Це дозволить перевірити правильність передавання параметрів, коректність обчислень і правильність повернення результатів.

Контрольні питання

1. Що називається функцією у програмуванні?
2. Яке призначення має прототип функції?
3. Чим відрізняється опис функції від її виклику?
4. Що таке формальні параметри функції?
5. Що таке фактичні параметри функції?
6. У чому полягає передавання параметрів за значенням?
7. У чому полягає передавання параметрів за адресою?
8. Яке призначення має оператор return?

Зміст звіту

1. Назва та мета роботи.
2. Короткі теоретичні відомості про функції та способи передавання параметрів.
3. Програмний код розроблених програм.
4. Результати виконання програм з різними наборами вхідних даних.
5. Висновки за результатами виконання роботи.

Практична робота 11. Декомпозиція задачі на підзадачі та модульна побудова програм

Ключові положення

Під час розробки програмного забезпечення однією з основних вимог є зрозуміла, логічна та зручна для супроводу структура програмного коду. Якщо складну задачу намагатися реалізувати у вигляді однієї великої послідовності інструкцій, така програма швидко стає важкою для аналізу, тестування та налагодження. Саме тому у програмуванні широко застосовується принцип декомпозиції задачі на підзадачі та модульної побудови програм.

Декомпозиція задачі полягає у розподілі складної задачі на кілька менших, логічно завершених підзадач. Кожна підзадача охоплює окремий етап загального алгоритму і може бути реалізована незалежно від інших частин програми. Такий підхід дозволяє поетапно проєктувати програму, спрощує аналіз алгоритму та дає змогу зосередитися на правильності реалізації кожного окремого компонента.

У межах модульного підходу кожна підзадача зазвичай реалізується у вигляді окремої функції. Функція має чітко визначене призначення, приймає вхідні дані через параметри, виконує необхідні операції та, за потреби, повертає результат. Таким чином, програма перетворюється на набір взаємопов’язаних модулів, кожен із яких виконує конкретну роль у загальному алгоритмі.

Модульна побудова програм є одним із базових принципів сучасного програмування. Вона забезпечує структурованість програмного коду, полегшує його повторне використання, спрощує тестування окремих частин програми та дозволяє швидше знаходити і виправляти помилки. Якщо програма складається з окремих функцій, правильність роботи можна перевіряти не лише для всієї програми, а й для кожного модуля окремо.

Під час декомпозиції задачі програміст спочатку аналізує її загальну структуру і визначає основні етапи розв'язання. Наприклад, типова прикладна задача може містити введення даних, перевірку коректності введеної інформації, виконання обчислень, аналіз результатів і виведення підсумкових даних. Кожен із цих етапів доцільно реалізувати окремим модулем. У результаті програма набуває чіткої внутрішньої структури.

Важливою перевагою модульного підходу є повторне використання коду. Якщо певна функція реалізує універсальну операцію, наприклад обчислення середнього значення, перевірку належності числа до інтервалу або виведення масиву на екран, її можна використати в різних частинах однієї програми або навіть у різних програмних проєктах. Це зменшує обсяг коду, скорочує час розробки та підвищує надійність програми, оскільки перевірені модулі не потрібно створювати повторно.

Модульна побудова програм безпосередньо пов'язана з використанням функцій. У мовах C і C++ програма починає виконання з функції main, але основна логіка алгоритму може бути розподілена між допоміжними функціями. Функція main у такому випадку виконує роль координатора: вона організовує виклики інших функцій, передає їм вхідні дані та отримує результати виконання.

Під час розробки модульної програми важливо визначити, які саме дані повинні передаватися між функціями. Якщо функція повинна лише виконати обчислення на основі вхідних параметрів і повернути результат, доцільно використовувати передавання параметрів за значенням. Якщо ж функція повинна змінити значення змінних, що існують у головній програмі, використовується передавання за адресою. Правильний вибір способу передавання параметрів є важливою частиною проєктування модульної програми.

Декомпозиція задачі має не лише програмну, а й алгоритмічну цінність. Коли задача поділяється на підзадачі, програміст краще розуміє її внутрішню логіку, легше визначає послідовність дій та простіше формулює алгоритм розв'язання. Такий підхід є особливо корисним під час розробки більших програм, де без чіткої структури швидко виникає плутанина у призначенні змінних, послідовності обчислень і взаємодії між частинами програми.

Окрему роль модульна побудова відіграє у тестуванні програмного забезпечення. Якщо програма складається з кількох незалежних функцій, кожну з них можна перевірити окремо на різних наборах вхідних даних. Це дає змогу швидше локалізувати помилку. Наприклад, якщо результат усієї програми є неправильним, але окремо функція введення даних працює правильно, перевірку можна зосередити на функції обчислення або функції аналізу результату.

Під час налагодження модульна структура також є перевагою. Якщо логіка програми розділена на невеликі підзадачі, програмісту легше зрозуміти, у якому саме місці виникла помилка. Крім того, короткі функції зазвичай легше читати та аналізувати, ніж один довгий фрагмент програмного коду.

У середовищі Visual Studio Code модульна побудова програм мовою C/C++ реалізується шляхом створення функцій у межах одного програмного файла або, у складніших проєктах, шляхом розміщення оголошень у заголовкових файлах, а реалізації – в

окремих файлах вихідного коду. У межах навчальної практичної роботи доцільно зосередитися на побудові програми у межах одного файла, але з чітким розподілом логіки на функції.

Таким чином, декомпозиція задачі на підзадачі та модульна побудова програм є важливими принципами програмування, які забезпечують логічну організацію алгоритму, підвищують зрозумілість програмного коду, полегшують повторне використання окремих фрагментів та спрощують тестування і налагодження програм.

Практичне завдання

1. Розробити програму, у якій задача обчислення середнього арифметичного трьох чисел поділена на окремі підзадачі: введення даних, обчислення результату, виведення результату.
2. Створити програму, у якій перевірка, чи є число парним, реалізована окремою функцією.
3. Розробити програму для знаходження більшого з двох чисел, у якій обчислення результату оформлено як окремий модуль.
4. Створити програму, у якій обчислення площі та периметра прямокутника реалізовано двома окремими функціями.
5. Розробити програму, у якій загальна задача обробки введених даних розділена на функції введення, перевірки, обчислення та виведення результату.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно проаналізувати умову задачі та визначити її основні етапи. На цьому етапі потрібно встановити, які підзадачі містить загальний алгоритм. Наприклад, окремими підзадачами можуть бути введення даних, перевірка правильності введення, виконання основного обчислення та виведення результату.

Після цього необхідно визначити, які підзадачі доцільно оформити у вигляді окремих функцій. Для кожної функції слід чітко сформулювати її призначення, визначити перелік вхідних параметрів і результат, який вона повинна повертати. Якщо функція не повинна повертати значення, для неї слід використовувати тип void.

У середовищі Visual Studio Code потрібно створити новий файл програмного коду з розширенням .c або .cpp. На початку файла слід підключити необхідні бібліотеки та оголосити прототипи всіх функцій, які використовуються у програмі. Це дасть змогу викликати функції у main незалежно від порядку їх опису у файлі.

Після цього необхідно реалізувати функцію main. У ній слід залишити лише загальну логіку керування програмою: виклик функцій введення даних, виклик функцій обчислення та виклик функцій виведення результатів. Основні обчислення не слід розміщувати безпосередньо у функції main, оскільки метою роботи є саме формування модульної структури програми.

Далі необхідно реалізувати всі допоміжні функції. Кожна функція повинна мати чітке призначення і виконувати лише одну логічно завершену дію. Не слід об’єднувати у межах однієї функції надто багато різних операцій, якщо їх можна природно розділити на кілька окремих підзадач.

Під час передавання даних між функціями необхідно правильно обрати спосіб передавання параметрів. Якщо функція лише використовує вхідні значення для обчислення результату, достатньо передавання за значенням. Якщо ж функція повинна змінити значення змінних у викликаючій частині програми, необхідно використовувати передавання за адресою.

Після реалізації програми необхідно виконати її компіляцію у Visual Studio Code. Якщо компілятор виявить помилки, слід перевірити правильність запису прототипів, заголовків функцій, кількості параметрів та типів результатів. Після виправлення помилок компіляцію потрібно повторити.

Після успішної компіляції програму необхідно протестувати на кількох наборах вхідних даних. Бажано перевірити правильність роботи не лише всієї програми в цілому, а й логіку окремих модулів. Це дозволить переконатися, що декомпозиція задачі виконана коректно, а всі функції правильно взаємодіють між собою.

Контрольні питання

1. Що таке декомпозиція задачі у програмуванні?
2. Яка мета поділу складної задачі на підзадачі?
3. Що розуміють під модульною побудовою програм?
4. Яку роль відіграють функції у модульній програмі?
5. Які переваги має модульна організація програмного коду?
6. Чому модульна структура полегшує тестування програм?
7. У чому полягає повторне використання програмного коду?
8. Які вимоги доцільно враховувати під час проєктування окремих модулів програми?

Зміст звіту

1. Назва та мета роботи.
2. Короткі теоретичні відомості про декомпозицію задачі та модульну побудову програм.
3. Опис підзадач, на які поділено загальний алгоритм.
4. Програмний код розроблених модулів і головної програми.
5. Результати виконання програми з різними наборами вхідних даних.
6. Висновки за результатами виконання роботи.

Практична робота 12. Тестування та налагодження багатомодульних програм

Ключові положення

Під час розробки програмного забезпечення недостатньо лише створити програму, яка формально відповідає поставленому завданню. Не менш важливим етапом є перевірка правильності її роботи, виявлення помилок та їх усунення. Саме для цього у програмуванні використовуються тестування та налагодження. Особливої уваги ці процеси потребують у багатомодульних програмах, де загальна задача поділена на кілька окремих функцій або програмних модулів, які взаємодіють між собою.

Багатомодульна програма має більш складну внутрішню структуру, ніж проста лінійна програма. Її робота залежить не лише від правильності реалізації кожного окремого модуля, а й від коректності передавання даних між функціями, правильності послідовності викликів, узгодженості типів параметрів і результатів, а також від логіки спільної роботи всіх компонентів. Саме тому тестування багатомодульних програм повинно проводитися системно і послідовно.

Тестуванням називають процес перевірки програми шляхом виконання її на різних наборах вхідних даних з подальшим аналізом отриманих результатів. Основною метою тестування є виявлення помилок, перевірка відповідності програми поставленому завданню та оцінювання її коректності в різних режимах роботи. Тестування не доводить абсолютну безпомилковість програми, але дозволяє встановити, чи правильно вона працює у типових, граничних і нестандартних ситуаціях.

У процесі створення програм зазвичай розрізняють кілька рівнів тестування. Для багатомодульних програм особливе значення мають модульне тестування та інтеграційне тестування. Модульне тестування передбачає перевірку окремих функцій або модулів незалежно від інших частин програми. Наприклад, якщо у програмі є окрема функція для обчислення суми елементів масиву, її доцільно перевірити окремо, використовуючи різні варіанти вхідних даних. Такий підхід дозволяє швидко виявити, чи правильно працює конкретний модуль.

Інтеграційне тестування виконується після перевірки окремих модулів і полягає у контролі правильності їх спільної роботи. На цьому етапі перевіряється, чи коректно передаються параметри між функціями, чи правильно використовуються результати виконання одних модулів в інших та чи забезпечує програма правильний кінцевий результат. У багатомодульних програмах інтеграційне тестування є особливо важливим, оскільки окремо правильні модулі можуть давати неправильний результат у разі некоректної взаємодії.

Під час тестування необхідно використовувати різні типи тестових даних. До них належать типові значення, граничні значення та помилкові або нестандартні вхідні дані. Типові значення відображають звичайний режим роботи програми. Граничні значення використовуються для перевірки поведінки програми на межах допустимого діапазону. Помилкові або нестандартні дані дозволяють оцінити стійкість програми до некоректного введення або неочікуваних ситуацій.

Наприклад, якщо функція обчислює середнє арифметичне кількох чисел, доцільно перевірити її на звичайних додатних значеннях, на нульових значеннях, на від’ємних числах, а також на граничних значеннях типу даних. Якщо програма виконує ділення, необхідно обов’язково перевірити випадки, коли знаменник може набувати нульового значення. Такий підхід дозволяє виявити не лише очевидні, а й приховані помилки алгоритму.

Після виявлення помилок виконується налагодження програми. Налагодженням називають процес пошуку причин помилок і внесення змін у програмний код з метою усунення цих помилок. Налагодження є невід’ємною частиною розробки програмного забезпечення, оскільки навіть правильно спроєктований алгоритм може бути реалізований із синтаксичними, логічними або семантичними помилками.

Синтаксичні помилки виникають у випадках порушення правил запису конструкцій мови програмування. До таких помилок належать пропущені символи, неправильний запис операторів, невідповідність дужок, неправильне оголошення функцій або змінних. Синтаксичні помилки зазвичай виявляються компілятором ще до запуску програми.

Семантичні помилки пов’язані з неправильним використанням конструкцій мови. Наприклад, це може бути передавання параметрів неправильного типу, використання неініціалізованих змінних, некоректний доступ до елементів масиву або неправильне повернення значення з функції. Такі помилки інколи виявляються під час компіляції, а інколи – лише під час виконання програми.

Логічні помилки є найбільш складними для виявлення. У цьому випадку програма успішно компілюється і запускається, однак працює неправильно з погляду поставленої задачі. Причиною може бути неправильна формула, помилкова умова, некоректна організація циклу, неправильна логіка викликів функцій або помилка у взаємодії модулів. Саме логічні помилки найчастіше виявляються під час тестування.

У багатомодульних програмах джерелом помилок часто є неправильна взаємодія між функціями. Наприклад, одна функція може повертати результат одного типу, а інша очікує інший тип даних. Або ж функція може змінювати значення параметра за адресою тоді, коли програміст очікував передавання за значенням. Можлива також ситуація, коли головна функція викликає модулі у неправильній послідовності, через що частина змінних ще не набуває потрібних значень на момент виконання обчислень.

Для налагодження програм у середовищі Visual Studio Code можна використовувати кілька підходів. Найпростішим є виведення проміжних результатів на екран. Такий спосіб дозволяє побачити, які значення мають змінні на різних етапах виконання програми, і встановити момент, коли обчислення починають відхилятися від правильного результату. Наприклад, можна тимчасово додати виведення параметрів на вході функції, результату обчислення в окремому модулі або значення змінної циклу під час виконання повторень.

Іншим важливим способом є покроковий аналіз програмного коду. Для цього у Visual Studio Code можуть використовуватися вбудовані засоби налагодження, які дозволяють виконувати програму поетапно, переглядати значення змінних, контролювати проходження через функції та аналізувати стек викликів. Особливо корисним є перегляд послідовності викликів функцій, оскільки у багатомодульних програмах помилки часто пов’язані саме з неправильною логікою переходів між модулями.

Ще одним важливим принципом налагодження є поетапна перевірка. Якщо програма складається з кількох модулів, не слід одразу шукати помилку в усій програмі. Доцільно спочатку перевірити правильність роботи кожного окремого модуля, а потім переходити до перевірки їх взаємодії. Такий підхід дає змогу локалізувати помилку та значно скоротити час її пошуку.

Під час тестування багатомодульних програм важливо також документувати результати перевірки. Для кожного тестового набору доцільно зазначати вхідні дані, очікуваний результат і фактичний результат виконання програми. Якщо результати не збігаються, це є підставою для подальшого налагодження. Такий підхід дозволяє зробити процес перевірки більш системним і обґрунтованим.

Значну роль відіграє і правильний стиль програмування. Якщо функції мають зрозумілі імена, модулі чітко відокремлені один від одного, а код містить логічні відступи та помірну кількість коментарів, тестування і налагодження виконуються значно простіше. Натомість хаотично написаний програмний код ускладнює аналіз і пошук помилок.

Таким чином, тестування та налагодження багатомодульних програм є важливими етапами розробки програмного забезпечення. Вони дозволяють перевірити правильність роботи окремих модулів, оцінити коректність їх взаємодії, виявити синтаксичні, семантичні

та логічні помилки, а також забезпечити надійність і працездатність усієї програмної системи.

Практичне завдання

1. Розробити багатомодульну програму, у якій окремими функціями реалізовано введення даних, основне обчислення та виведення результату, і виконати її модульне тестування.
2. Створити програму з кількох функцій, одна з яких навмисно містить логічну помилку, та виконати її налагодження.
3. Розробити програму, у якій одна функція обчислює значення виразу, друга перевіряє коректність вхідних даних, а третя виводить результат, після чого провести інтеграційне тестування.
4. Виконати перевірку багатомодульної програми на типових, граничних і помилкових вхідних даних.
5. Підготувати таблицю тестових прикладів для розробленої програми та проаналізувати результати її виконання.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно підготувати у Visual Studio Code програму, що складається з кількох окремих функцій. Доцільно, щоб одна функція відповідала за введення даних, інша – за основні обчислення, а ще одна – за виведення результату. За потреби можна додати окрему функцію для перевірки коректності вхідних даних.

Після створення програми необхідно перевірити правильність оголошення прототипів функцій, відповідність типів параметрів і правильність використання оператора return. Особливу увагу слід звернути на узгодженість типів між викликом функції та її описом.

Далі потрібно виконати модульне тестування. Для цього слід по черзі перевірити кожну функцію на окремих наборах даних. Якщо функція виконує обчислення, необхідно підготувати такі значення параметрів, для яких правильний результат можна заздалегідь визначити вручну. Це дозволить легко встановити правильність її роботи.

Після перевірки окремих функцій слід перейти до інтеграційного тестування, тобто перевірки всієї програми в цілому. На цьому етапі необхідно переконатися, що дані правильно передаються між функціями, результати обчислень використовуються коректно, а загальна послідовність викликів модулів відповідає алгоритму розв’язання задачі.

У процесі тестування необхідно використовувати різні типи вхідних даних. Слід перевірити типові значення, граничні значення, а також некоректні або неочікувані дані. Якщо програма не містить перевірки коректності введення, доцільно додати відповідний модуль або передбачити обробку таких ситуацій.

Якщо під час тестування буде виявлено неправильний результат, необхідно перейти до налагодження програми. Найпростішим способом є виведення проміжних значень змінних на екран у різних точках виконання програми. Це дозволяє встановити, у якому саме модулі або на якому етапі обчислення виникає відхилення від очікуваного результату.

За наявності відповідного налаштування у Visual Studio Code доцільно використати засоби покрокового налагодження. Під час такого аналізу потрібно простежити

послідовність викликів функцій, значення параметрів на вході та виході, а також правильність змін локальних змінних у тілі функцій.

Після усунення помилок необхідно повторно виконати тестування програми на тих самих наборах даних. Це дозволить переконатися, що виявлені недоліки усунуті, а внесені зміни не призвели до появи нових помилок.

Контрольні питання

1. Що таке тестування програмного забезпечення?
2. Яка мета модульного тестування?
3. У чому полягає інтеграційне тестування?
4. Які типи помилок можуть виникати у програмі?
5. Чим логічна помилка відрізняється від синтаксичної?
6. Що таке налагодження програми?
7. Чому у багатомодульних програмах важливо перевіряти взаємодію між функціями?
8. Які способи виявлення помилок доцільно використовувати у Visual Studio Code?

Зміст звіту

1. Назва та мета роботи.
2. Короткі теоретичні відомості про тестування і налагодження багатомодульних програм.
3. Структура розробленої програми та опис її модулів.
4. Таблиця тестових прикладів із вхідними даними, очікуваними та фактичними результатами.
5. Опис виявлених помилок і способів їх усунення.
6. Програмний код налагодженої програми.
7. Висновки за результатами виконання роботи.

Практична робота 13. Реалізація алгоритмів обробки одновимірних масивів

Ключові положення

Під час розробки програмного забезпечення часто виникає потреба працювати не з окремими значеннями, а з наборами однотипних даних. Якщо кількість таких даних є більшою за кілька елементів, використання окремих змінних для кожного значення стає незручним і призводить до ускладнення структури програми. Для розв’язання цієї проблеми у програмуванні використовуються масиви. Масив є структурою даних, що дає змогу зберігати послідовність елементів одного типу під спільним іменем. Доступ до кожного елемента масиву здійснюється за допомогою індексу.

Одновимірний масив можна розглядати як впорядковану послідовність однотипних значень, розміщених у суміжних комірках пам’яті. У мовах C і C++ нумерація елементів масиву починається з нуля. Це означає, що перший елемент має індекс 0, другий – індекс 1, а останній – індекс, що на одиницю менший за розмір масиву. Такий спосіб організації даних дозволяє ефективно виконувати різні операції обробки, зокрема введення, виведення, пошук,

обчислення сум, добутків, середніх значень, визначення максимального або мінімального елемента.

Алгоритми обробки одновимірних масивів ґрунтуються на послідовному перегляді їх елементів. Для цього зазвичай використовуються цикли. Найчастіше застосовується цикл for, оскільки кількість елементів масиву, як правило, відома або задається до початку обробки. Під час кожної ітерації циклу програма звертається до чергового елемента масиву та виконує необхідні операції.

Однією з базових задач є введення елементів масиву. У програмі це реалізується циклом, у межах якого для кожного індексу зчитується значення відповідного елемента. Аналогічно виконується виведення елементів масиву. Цикл дозволяє послідовно відображати значення всіх елементів, що забезпечує наочне представлення вмісту масиву.

Окрім введення та виведення, важливими задачами є обчислення характеристик масиву. Наприклад, для знаходження суми всіх елементів використовується змінна-накопичувач, яка на початку ініціалізується нулем. Далі під час проходження по масиву значення кожного елемента додається до накопичувача. Після завершення циклу змінна містить загальну суму. Аналогічно можна обчислювати добуток елементів, середнє арифметичне, суму лише додатних або лише парних елементів.

Ще однією типовою задачею є пошук максимального або мінімального значення. Для цього на початку як поточний максимум або мінімум доцільно взяти перший елемент масиву. Далі всі інші елементи послідовно порівнюються з поточним значенням, і за потреби змінна максимуму або мінімуму оновлюється. Такий алгоритм є простим і водночас ефективним для одновимірних масивів.

Під час обробки масивів часто виникає необхідність не лише визначити значення найбільшого або найменшого елемента, а й знайти його позицію в масиві. У такому випадку, окрім змінної для збереження значення, використовується ще одна змінна для збереження індексу відповідного елемента. Це дозволяє після завершення обробки вивести не тільки саме значення, а й номер його розташування у масиві.

Ще одним поширеним видом обробки є підрахунок кількості елементів, що задовольняють певну умову. Наприклад, програма може визначати кількість додатних, від’ємних, нульових, парних або непарних елементів. У таких задачах використовується змінна-лічильник, яка збільшується на одиницю кожного разу, коли черговий елемент масиву задовольняє задану умову.

У задачах обробки масивів важливу роль відіграє правильна організація індексів. Індекс повинен залишатися у межах допустимого діапазону. Якщо програма звертається до елемента з індексом, що виходить за межі масиву, виникає помилка доступу до пам’яті, яка може призвести до некоректної роботи програми або аварійного завершення. Саме тому під час програмування необхідно уважно контролювати межі зміни змінної циклу.

Для реалізації алгоритмів обробки одновимірних масивів у межах дисципліни використовується середовище Visual Studio Code та мова програмування C/C++. У Visual Studio Code створюється файл із розширенням .c або .cpp, після чого програмний код компілюється за допомогою встановленого компілятора. Такий підхід дозволяє не лише реалізувати алгоритм, а й одразу перевірити правильність його роботи на конкретних наборах даних.

Під час розробки програм для роботи з масивами важливо дотримуватися модульного підходу. Наприклад, доцільно окремо реалізувати введення елементів, окремо – їх виведення, а також окремі функції для обчислення суми, пошуку максимального елемента чи підрахунку

кількості значень, що задовольняють умову. Такий підхід полегшує структуру програми, спрощує тестування і дає змогу повторно використовувати окремі фрагменти коду.

Таким чином, алгоритми обробки одновимірних масивів є важливою складовою підготовки з алгоритмізації та програмування. Вони формують навички послідовної обробки наборів даних, організації циклів, використання змінних-накопичувачів і лічильників, а також навички аналізу значень елементів масиву за різними критеріями.

Практичне завдання

1. Розробити програму для введення та виведення елементів одновимірного масиву.
2. Створити програму для обчислення суми всіх елементів масиву.
3. Розробити програму для знаходження найбільшого та найменшого елементів масиву.
4. Створити програму для обчислення кількості додатних, від'ємних і нульових елементів масиву.
5. Розробити програму для обчислення середнього арифметичного елементів одновимірного масиву.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно у середовищі Visual Studio Code створити новий файл програмного коду з розширенням .c або .cpp. Після цього слід підключити необхідні бібліотеки та підготувати структуру програми з функцією main.

На початковому етапі необхідно оголосити одновимірний масив відповідного типу, а також змінні, що використовуватимуться для зберігання кількості елементів, суми, середнього значення, максимального та мінімального елементів, а також лічильників для підрахунку елементів за певними ознаками. Якщо кількість елементів задається користувачем, доцільно спочатку зчитати це значення, а потім виконувати обробку у межах відповідного діапазону індексів.

Далі необхідно реалізувати введення елементів масиву. Для цього використовується цикл, у межах якого для кожного індексу виконується зчитування значення з клавіатури. Бажано, щоб програма повідомляла користувача, який саме елемент потрібно ввести, наприклад за номером позиції.

Після введення елементів доцільно реалізувати їх виведення на екран, щоб перевірити правильність заповнення масиву. Для цього також використовується цикл. Якщо потрібно, виведення можна організувати в один рядок або у вигляді стовпця, залежно від умов завдання.

Під час реалізації задачі знаходження суми всіх елементів необхідно використати змінну-накопичувач, якій перед початком циклу присвоюється початкове значення нуль. Далі у циклі до неї додається значення кожного елемента масиву. Після завершення циклу результат виводиться на екран.

Для знаходження найбільшого та найменшого елементів доцільно початково присвоїти змінним максимуму і мінімуму значення першого елемента масиву. Далі у циклі порівнювати кожен наступний елемент з поточними значеннями максимуму і мінімуму та за потреби оновлювати їх.

Під час підрахунку кількості додатних, від'ємних і нульових елементів необхідно використати окремі лічильники, які перед початком обробки ініціалізуються нулем. Далі у процесі перегляду масиву для кожного елемента виконується перевірка його значення, і відповідний лічильник збільшується на одиницю.

Для обчислення середнього арифметичного елементів масиву спочатку потрібно знайти суму всіх елементів, а потім поділити її на кількість елементів. Під час цього слід врахувати типи даних. Якщо масив має цілочисловий тип, для отримання точного результату доцільно виконати перетворення типів, щоб уникнути цілочислового ділення.

Після написання програмного коду необхідно виконати його компіляцію. Якщо компілятор виявить помилки, слід перевірити правильність оголошення масиву, межі циклів, типи змінних, а також правильність звернення до елементів за індексами. Після виправлення помилок програму потрібно повторно скомпілювати і запустити.

Після успішного запуску слід протестувати програму на кількох наборах вхідних даних. Доцільно використовувати як типові значення, так і набори, що містять додатні, від'ємні та нульові елементи. Це дозволить перевірити правильність роботи всіх реалізованих алгоритмів.

Контрольні питання

1. Що називається одновимірним масивом?
2. Як здійснюється доступ до елементів масиву у мовах C і C++?
3. З якого значення починається індексація елементів масиву?
4. Які задачі обробки одновимірних масивів є типовими?
5. Як обчислюється сума елементів масиву?
6. Яким чином визначають найбільший і найменший елементи масиву?
7. Для чого використовуються змінні-лічильники під час обробки масивів?
8. Які помилки можуть виникати під час звернення до елементів масиву?

Зміст звіту

1. Назва та мета роботи.
2. Короткі теоретичні відомості про одновимірні масиви та алгоритми їх обробки.
3. Програмний код розроблених програм.
4. Результати виконання програм на різних наборах вхідних даних.
5. Аналіз отриманих результатів.
6. Висновки за результатами виконання роботи.

Практична робота 14. Реалізація алгоритмів пошуку та сортування даних

Ключові положення

Під час обробки масивів і послідовностей даних одними з найважливіших задач є пошук потрібного елемента та впорядкування елементів за певним правилом. Такі задачі виникають у більшості прикладних програм, пов'язаних із числовими розрахунками, статистичною обробкою, формуванням звітів, аналізом результатів вимірювань і роботою з

довідковою інформацією. Саме тому алгоритми пошуку та сортування належать до базових алгоритмів інформатики та програмування.

Пошук даних у найпростішому випадку полягає у визначенні, чи міститься в масиві елемент із заданим значенням, а також у знаходженні його позиції. Якщо дані ще не впорядковані, найпростішим способом розв'язання такої задачі є лінійний пошук. Його суть полягає у послідовному перегляді всіх елементів масиву, починаючи з першого. На кожному кроці виконується порівняння поточного елемента із шуканим значенням. Якщо значення збігаються, пошук завершується, і програма фіксує індекс знайденого елемента. Якщо переглянуто весь масив, а збіг не виявлено, робиться висновок про відсутність шуканого елемента.

Перевагою лінійного пошуку є простота реалізації. Для його використання не потрібно попередньо впорядковувати дані, тому цей алгоритм зручний у навчальних задачах і в тих випадках, коли кількість елементів невелика. Водночас ефективність такого пошуку обмежена, оскільки у найгіршому випадку доводиться перевіряти всі елементи масиву. Якщо масив має великий розмір, час виконання лінійного пошуку суттєво зростає.

У багатьох практичних задачах важливим є не лише пошук конкретного значення, а й пошук елемента за певною умовою. Наприклад, програма може шукати перший додатний елемент, найбільший елемент, елемент, що перевищує певне значення, або кількість елементів, які задовольняють визначену умову. Усі ці задачі також реалізуються шляхом послідовного перегляду масиву, але критерій перевірки змінюється залежно від умови задачі.

Ще однією фундаментальною задачею є сортування. Сортуванням називають процес впорядкування елементів масиву за зростанням, спаданням або за іншим визначеним правилом. Відсортовані дані зручніше аналізувати, відображати користувачу та використовувати у подальших обчисленнях. Крім того, багато ефективніших алгоритмів пошуку можуть застосовуватися лише до вже впорядкованих даних. Саме тому сортування є важливою підготовчою операцією перед виконанням складніших алгоритмів обробки.

У початковому курсі програмування зазвичай розглядаються прості алгоритми сортування, які легко зрозуміти і реалізувати у програмному коді. До таких алгоритмів належать сортування обміном, сортування вибором і сортування вставками. Хоча вони не є найефективнішими для великих масивів, саме на їх прикладі зручно формувати уявлення про принципи впорядкування даних.

Сортування обміном, яке часто називають бульбашковим сортуванням, ґрунтується на багаторазовому порівнянні сусідніх елементів масиву. Якщо два сусідні елементи розміщені у неправильному порядку, вони міняються місцями. Після одного повного проходу найбільший або найменший елемент, залежно від способу сортування, переміщується у свою кінцеву позицію. Повторюючи такі проходи, можна поступово впорядкувати весь масив. Цей алгоритм є простим для розуміння, але потребує значної кількості порівнянь та перестановок.

Сортування вибором базується на пошуку найменшого або найбільшого елемента у невідсортованій частині масиву та обміні його з елементом, який повинен стояти на поточній позиції. На першому етапі знаходиться найменший елемент у всьому масиві і переноситься на перше місце. Далі шукається найменший елемент серед решти елементів і розміщується на другій позиції. Процес повторюється до повного впорядкування масиву. Такий алгоритм виконує менше обмінів, ніж бульбашкове сортування, але також потребує значної кількості порівнянь.

Сортування вставками реалізує інший підхід. Спочатку вважається, що перший елемент уже впорядкований. Далі кожен наступний елемент по черзі вставляється у правильне місце серед уже впорядкованої частини масиву. Такий спосіб нагадує впорядкування карт у руці, коли нова карта вставляється на відповідну позицію серед уже розташованих карт. Цей алгоритм зручний у випадках, коли масив уже частково впорядкований, оскільки тоді кількість перестановок може бути меншою.

Під час реалізації алгоритмів сортування особливу увагу слід приділяти обміну значень елементів масиву. Для перестановки двох елементів зазвичай використовується допоміжна змінна, у якій тимчасово зберігається одне зі значень. Це дозволяє виконати обмін коректно і не втратити жодного з елементів.

Алгоритми пошуку і сортування доцільно оцінювати не лише з погляду правильності, а й з погляду ефективності. Ефективність алгоритму визначається кількістю операцій, які потрібно виконати для досягнення результату. У випадку пошуку основними операціями є порівняння значень. У випадку сортування – порівняння і перестановки елементів.

Лінійний пошук у найгіршому випадку вимагає перегляду всіх елементів масиву, тому кількість порівнянь зростає пропорційно кількості елементів. Прості алгоритми сортування працюють повільніше. Для масиву з n елементів кількість операцій у них приблизно пропорційна квадрату кількості елементів. Це означає, що зі збільшенням розміру масиву час виконання таких алгоритмів зростає досить швидко. Саме тому для великих обсягів даних у практичному програмуванні використовують ефективніші алгоритми сортування, однак на початковому етапі вивчення достатньо оволодіти базовими методами.

У межах цієї практичної роботи програмування виконується у середовищі Visual Studio Code мовою C/C++. Студент повинен не лише реалізувати алгоритми пошуку і сортування, а й перевірити їх роботу на різних наборах даних, проаналізувати правильність одержаних результатів і звернути увагу на відмінності в логіці роботи кожного алгоритму. Така робота формує практичні навички створення програм для обробки масивів і закладає основу для подальшого вивчення складніших алгоритмів.

Практичне завдання

1. Розробити програму для виконання лінійного пошуку заданого елемента в одновимірному масиві.
2. Створити програму для знаходження індексу першого елемента масиву, що дорівнює заданому значенню.
3. Розробити програму сортування одновимірного масиву за зростанням методом обміну.
4. Створити програму сортування одновимірного масиву за спаданням методом вибору.
5. Розробити програму, у якій виконуються пошук елемента в масиві до сортування і після сортування, з подальшим порівнянням результатів.

Методичні вказівки до виконання практичного завдання

Перед початком виконання роботи необхідно у середовищі Visual Studio Code створити новий файл програмного коду з розширенням .c або .cpp. На початку програми слід

підключити необхідні бібліотеки, після чого реалізувати функцію main та, за потреби, допоміжні функції для введення масиву, його виведення, пошуку елементів і сортування.

На першому етапі необхідно організувати введення масиву. Для цього користувач повинен задати кількість елементів, після чого у циклі вводяться значення всіх елементів масиву. Доцільно одразу реалізувати також виведення масиву на екран, щоб перевірити правильність його формування.

Під час реалізації лінійного пошуку необхідно організувати послідовний перегляд елементів масиву. У циклі слід порівнювати кожен елемент із шуканим значенням. Якщо збіг виявлено, необхідно зафіксувати індекс елемента і завершити пошук. Якщо після завершення циклу збігу немає, програма повинна вивести повідомлення про відсутність шуканого елемента.

Для задачі пошуку першого входження значення необхідно зупинити виконання циклу одразу після першого знайденого збігу. Це дозволить правильно визначити саме першу позицію шуканого значення, навіть якщо воно повторюється у масиві кілька разів.

Під час реалізації сортування обміном необхідно використати вкладені цикли. У внутрішньому циклі виконуються порівняння сусідніх елементів і, за потреби, їх перестановка. Для обміну елементів слід застосовувати допоміжну змінну. Після завершення сортування масив необхідно вивести на екран.

Для реалізації сортування вибором також використовуються вкладені цикли. Для кожної позиції масиву потрібно знайти елемент з потрібною властивістю, наприклад найбільший для сортування за спаданням, і виконати його обмін з елементом поточної позиції. Після кожного проходу одна позиція вважається остаточно впорядкованою.

Під час виконання завдання з порівняння пошуку до сортування і після сортування необхідно звернути увагу на те, що сам алгоритм лінійного пошуку не змінюється. Однак після сортування структура масиву стає впорядкованою, що полегшує подальший аналіз даних. У висновках доцільно зазначити, що просте сортування є корисним не лише для візуального впорядкування, а й як підготовчий етап до застосування складніших алгоритмів.

Після написання програмного коду необхідно виконати компіляцію. Якщо компілятор виявить помилки, слід перевірити правильність оголошення змінних, межі циклів, логіку порівнянь, а також правильність обміну елементів. Після виправлення помилок програму потрібно повторно скомпілювати і протестувати.

Тестування слід виконувати на кількох наборах вхідних даних. Доцільно перевірити масиви з різною кількістю елементів, у тому числі з уже впорядкованими даними, з повторюваними елементами, з додатними і від’ємними числами. Це дозволить переконатися у правильності реалізації алгоритмів пошуку та сортування.

Контрольні питання

1. Що називається лінійним пошуком?
2. У чому полягає принцип роботи лінійного пошуку?
3. Що називається сортуванням масиву?
4. У чому полягає принцип роботи сортування обміном?
5. У чому полягає принцип роботи сортування вибором?
6. Для чого під час сортування використовується допоміжна змінна?
7. Як оцінюють ефективність алгоритмів пошуку та сортування?
8. Чому прості алгоритми сортування є малоефективними для великих масивів?

Зміст звіту

1. Назва та мета роботи.
2. Короткі теоретичні відомості про алгоритми пошуку та сортування даних.
3. Програмний код реалізованих алгоритмів.
4. Результати виконання програм на різних наборах вхідних даних.
5. Порівняльний аналіз роботи алгоритмів пошуку та сортування.
6. Висновки за результатами виконання роботи.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Основна

1. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. 4th ed. Cambridge, MA : MIT Press, 2022. 1312 p. ISBN 9780262046305.
2. Седжвік Р., Уейн К. Алгоритми. 4-те вид. Київ : BHV, 2018. 992 с. ISBN 9789665524762.
3. Рудий Т. В., Паранчук Я. С., Сеник В. В. Алгоритмізація та програмування. Ч. 1. Структурне програмування : навч. посіб. Львів : ЛДУВС, 2023. 240 с.
4. Ярошко С. А., Ярошко О. С. Методи розробки алгоритмів. Програмування : навч. посіб. Львів : ЛНУ імені Івана Франка, 2022. 248 с.
5. Лавріщева К. М. Основи алгоритмізації та програмування. Київ : Академія, 2020. 304 с.

Допоміжна

6. Вірт Н. Алгоритми + структури даних = програми. Київ : Діалектика, 2019. 366 с.
7. Трофименко О. Г., Прокоп Ю. В., Логінова Н. І., Задерейко О. В. С++. Алгоритмізація та програмування : підручник. 2-ге вид. Одеса : Фенікс, 2019. 477 с.
8. Prokop Y. V., Bukata L. N., Trofymenko O. G. Algorithmization and Programming. Structured Data Programming. Odesa : A. S. Popov ONAT, 2020. 57 p.
9. ISO/IEC/IEEE 12207:2017. Systems and software engineering – Software life cycle processes. Geneva : ISO, 2017.

Інформаційні ресурси

10. ISO. Software life cycle processes (ISO/IEC/IEEE 12207:2017). URL: https://www.iso.org/standard/63712.html
11. Microsoft Learn. C++ documentation. URL: https://learn.microsoft.com/cpp
12. cppreference.com. C++ language reference. URL: https://en.cppreference.com
13. MIT OpenCourseWare. Introduction to Programming. URL: https://ocw.mit.edu
14. Algorithms Specialization (Coursera). URL: https://www.coursera.org/specializations/algorithms

Навчальне видання

Русу Олександр Петрович

АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМУВАННЯ

Методичні вказівки до виконання практичних робіт
для здобувачів вищої освіти, які навчаються за спеціальностями
галузі «Інформаційні технології»

Українською мовою