О. Г. Трофименко, Ю. В. Прокоп, О. Г. Янковський

Структури даних: практикум

Навчально-методичний посібник
для підготовки здобувачів вищої освіти
галузі знань 12 «Інформаційні технології»

Одеса
Фенікс
2022

УДК 004.43: 004.421.2 (076)
Т 76

Рекомендовано навчально-методичною радою
Національного університету «Одеська юридична академія»
(протокол № 3 від 17 травня 2022 р.)

Рецензенти:
Малаксіано М. О. – доктор технічних наук, професор, завідувач кафедри «Технічна кібернетика й інформаційні технології ім. проф. Р. В. Меркта» Одеського національного морського університету;

Панченко Б. Є. – доктор фізико-математичних наук, с.н.с., в.о. завідувача кафедри «Інженерія програмного забезпечення» Державного університету інтелектуальних технологій і зв'язку

Трофименко О. Г., Прокоп Ю. В., Янковський О.Г.

Т 76 Структури даних: практикум : навч.-метод. посібник [Електронне видання] / О. Г. Трофименко, Ю. В. Прокоп, О. Г. Янковський. – Одеса : Фенікс, 2022. – 113 с.

ISBN 978-966-928-807-3

Розглянуто основні засоби програмного створення і роботи з структурами даних, зокрема лінійними списками, стеками, чергами, деревами, графами. Містить індивідуальні завдання до лабораторних занять та самостійної роботи в межах навчальних дисциплін, що вивчають структури даних. На численних прикладах розглянуто специфіку створення структур даних, пошуку і відбору певних даних, різноманітного маніпулюваннями даними. Призначено для студентів з метою закріплення лекційного матеріалу і підготовки до лабораторних занять та самостійної роботи з дисциплін, що вивчають структури даних.

Для підготовки здобувачів вищої освіти галузі знань 12 «Інформаційні технології».

УДК 004.43: 004.421.2 (076)

ISBN 978-966-928-807-3

ЗМІСТ

Передмова

У навчально-методичному посібнику подано короткий теоретичний матеріал та численні наочні приклади програмної реалізації засобами мови С++ формування та опрацювання різних структур даних: лінійних списків (односпрямованих, двоспрямованих, циклічних), стеків, черг, дерев, графів, автоматів тощо. Запропоновано до виконання сім лабораторних та три самостійні роботи, кожна з яких має по двадцять п'ять індивідуальних варіантів завдань. Деякі з лабораторних робіт мають різнорівневі групи завдань і розраховані на декілька аудиторних занять. У подальшому при оцінюванні знань викладач може враховувати рівень складності виконання роботи та оптимальність алгоритму програми виконаної роботи.

Перед виконанням лабораторного завдання студентові потрібно:

- уточнити у викладача індивідуальне завдання;
- вивчити відповідні розділи теоретичного курсу згідно з лекційними записами та навчальною літературою;
- розробити мовою С++ програми розв'язання задач;
- підготувати протокол виконання лабораторної роботи і подати його викладачеві для перевірки.

До виконання лабораторної роботи студент має самостійно ознайомитись з теоретичними відомостями та відповісти на контрольні запитання. Результатом виконання роботи є протокол.

Зміст протоколу лабораторної роботи має такі складові:

- тема і мета лабораторної роботи;
- відповіді на контрольні запитання;
- схематичне зображення відповідних структур даних;
- тексти програм мовою С++;
- результати обчислень на комп'ютері;
- висновки.

Правильність роботи програми та здобутих результатів перевіряються і оцінюються викладачем.

Структура дисципліни

Навчальні дисципліни, присвячені вивченню структур даних, можуть мати різну назву (наприклад «Алгоритми і структури даних» або «Структури даних», «Динамічні структури даних тощо). Матеріал щодо програмних засобів роботи зі структурами даних вивчається здобувачами вищої освіти в університетах України, що навчаються за освітньо-кваліфікаційним рівнем «бакалавр» галузі знань 12 «Інформаційні технології», переважно на другому курсі навчання бакалаврату спеціальностей 121, 122 та інших. У західних вишах такі змістові розділи здебільшого входять у курс CS2. Специфіка організації структур даних, як необхідних атрибутів програмних конструкцій, використання технології організації пам'яті під певні сучасні базові структури даних (автоматні структури, прості спискові структури і складні нелінійні структури даних), володіння навиками розробки лінійних та нелінійних структур даних засобами алгоритмічних мов разом з іншими складають теоретично-практичну основу сукупності компетентностей, що формують профіль фахівця в галузі інформаційних технологій.

Метою вивчення специфіки роботи зі структурами даних є формування умінь ефективно організувати дані і писати максимально продуктивний програмний код.

Предметом вивчення є методи розроблення і програмного конструювання структур даних.

Підсумкові результати навчання навчальної дисципліни деталізують такі програмні **результати навчання**:

– знати опис даних на абстрактному, логічному та фізичному рівні, специфіку організації структур даних, як необхідних атрибутів програмних конструкцій;

– використовувати технології організації пам'яті під певні сучасні базові структури даних: автоматні структури, прості спискові структури і складні нелінійні структури даних;

– володіти навиками розробки лінійних та нелінійних структур даних засобами алгоритмічних мов;

– застосовувати на практиці ефективні підходи щодо проектування програмного забезпечення.

Перелік тем лабораторних робіт

Лабораторна робота № 1. Лінійні списки: створення, перегляд.

Лабораторна робота № 2. Лінійні списки: вставлення і видалення елементів.

Лабораторна робота № 3. Двозв'язні лінійні списки.

Лабораторна робота № 4. Циклічні однозв’язні списки.

Лабораторна робота № 5. Черга і стек.

Лабораторна робота № 6. Бінарні дерева. Алгоритми пошуку на деревах.

Лабораторна робота № 7. Графи.

Кожна із запропонованих до виконання робіт має теоретичні відомості з прикладами програмної реалізації та контрольні запитання для закріплення відповідного тематичного матеріалу.

Перелік тем для самостійної роботи

Самостійна робота № 1. Циклічні двозв’язні списки.

Самостійна робота № 2. Бінарні дерева.

Самостійна робота № 3. Автомати.

Правильність виконаних лабораторних та самостійних робіт перевіряє викладач.

Лабораторна робота 1

Лінійні списки: створення, перегляд

Теоретичні відомості

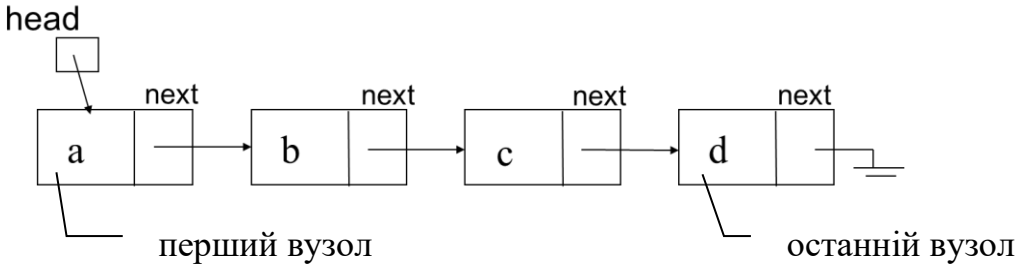
Список – лінійна послідовність зі змінного числа елементів, кожен з яких містить вказівники (посилається) на своїх сусідів. Зв'язні списки є динамічними, тому довжина списку може збільшуватися або зменшуватися за потреби. **Довжина списку** дорівнює числу елементів у списку, список нульової довжини називається порожнім списком.

Зв'язні списки є доречними, коли кількість елементів у структурі даних наперед невідома.

Списки є засобом організації структури даних, де елементи деякого типу утворюють ланцюжок. Для зв'язування елементів у списку використовують систему вказівників.

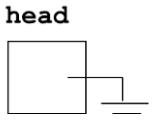
Розрізняють односпрямовані (однозв'язні), двоспрямовані (двозв’язні) і циклічні лінійні списки. Найбільш простою динамічною структурою є односпрямований список, елементами якого служать об'єкти структурного типу.

Структура односпрямованого зв'язаного лінійного списку:



Кожен елемент списку не обов'язково фізично слідує в пам’яті за попереднім. У зв'язаному списку порядок елементів визначається не номерами, як у масиві, а вказівниками, що входять до складу елементів списку. Зв'язні списки можна підтримувати у впорядкованому вигляді.

Вказівник початку списку (голови списку) вказує на перший вузол у зв'язаному списку. Якщо head є NULL, зв'язний список порожній.

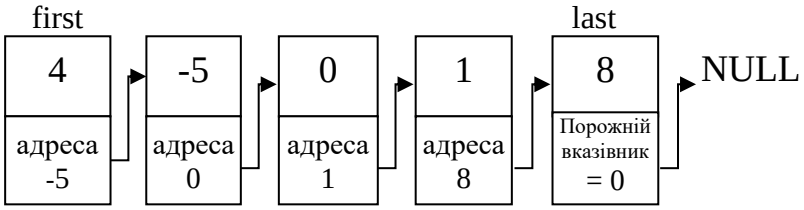


Порожній список є вказівником зі значенням NULL:

```
head = NULL;
```

Значення NULL у полі вказівника останнього елемента списку свідчить про кінець списку.

Односпрямований (однозв'язний) список − це структура даних, що утворює послідовність елементів, у кожному з яких зберігається значення і вказівник на подальший елемент списку. В останньому елементі вказівник на подальший елемент дорівнює NULL.



Опис найпростішого елемента такого списку виглядає так:

```
struct ім'я_типу
{
    інформаційне поле;
    адресне поле;
}
```

де `інформаційне поле` – це поле будь-якого, раніше оголошеного або стандартного типу; `адресне поле` – це вказівник на об'єкт того самого типу, що і визначена структура (в нього записується адреса подальшого елемента списку).

Наприклад:

```
struct Node
{
    int data; //інформаційне поле
    Node *next;  //адресне поле
};
```

Інформаційних полів може бути декілька, наприклад:

```
struct point
{
        char *name;                 //інформаційне поле
        int age;                    //інформаційне поле
        point *next;                //адресне поле
};
```

Елемент розміщується в пам'яті динамічно:

```
Node *cur;
cur = new Node;
```

При роботі з елементами списку, наприклад, при виведенні вмісту зв'язаного списку, звертатимемось до полів елемента за допомогою оператора **->** (замість оператора доступу "**.**"):

```
cur->data;
```

Під час операції проходу списком відвідується кожний елемент списку. Вказівникова змінна cur вказує на поточний вузол списку:

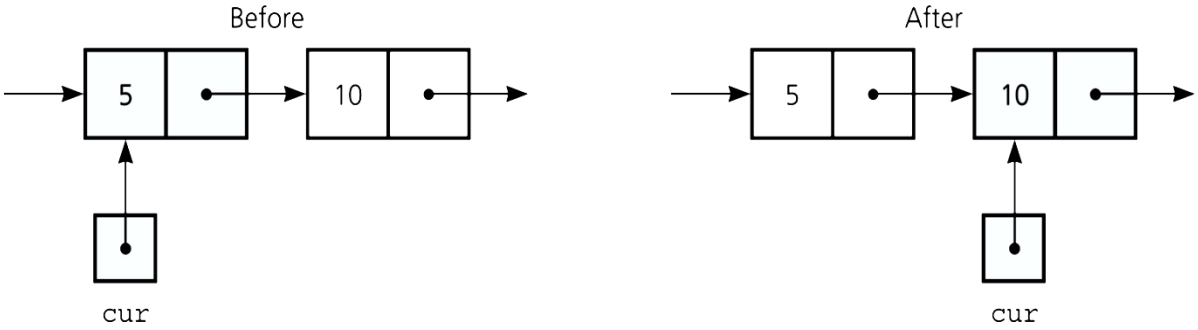
```
for (Node *cur = head; cur != NULL; cur = cur->next)
    cout << cur->data << endl;
```

Те ж саме за допомогою циклу while:

```
Node *cur = head;
while (cur != NULL)
{
        cout << cur->data << endl;
```

```
        cur = cur->next;
}
```

Результат присвоєння cur = cur->next:



Базові операції зі списком:

- створення списку;
- виведення (перегляд) списку;
- прохід списком;
- пошук елемента у списку;
- перевірка відсутності елементів у списку (NULL);
- долучення (вставлення) вузла;
- вилучення вузла;
- видалення всього списку.

Слід звернути увагу на те, що перед виконанням будь-яких операцій з лінійним односпрямованим списком треба забезпечити позиціонування вказівника на перший елемент. Інакше частина або весь список буде недоступний.

Наприклад:

```
struct Single_List          // структура даних
{
  int Data;                 // інформаційне поле
  Single_List *Next;        // адресне поле
};
. . . . . .
Single_List *Head;          // вказівник на перший елемент списку
. . . . . .
// вказівник на поточний елемент списку (за потреби)
Single_List *Current;
```

При налагодженні програм, що працюють зі списками, можуть виникати специфічні помилки, пов'язані з некоректним переустановленням зв'язків між елементами:

- втрата зв'язності – унаслідок порушення порядку присвоювання (переустановлення зв'язків) деякі елементи списку можуть опинитися недоступними або «втраченими»;
- топологічні помилки – порушується лінійність списку, «список перестає бути списком», а стає топологічно більш складною структурою, на якій поведінка програми стає непередбачуваною.

Аналогічний ефект можна отримати, якщо забути скорегувати заголовок списку, коли операція стосується першого елемента. Наприклад, якщо в сортуванні однозв'язного списку вставками забути присвоювання вказівника при вставці перед першим, то в такій ситуації черговий елемент вхідного списку просто буде втрачено. Цей ефект буде «мерехтливим»: губитися будуть деякі елементи зі значенням, меншим за всі попередні у вхідному. Але програма все ж працюватиме. У випадках більш серйозного порушення топології вона може зациклитися або взагалі «лягти».

Приклади програм

Приклад 1. Вводити числа у список, доки не введено число -1000, вивести список на екран.

```
#include<iostream>
using namespace std;
struct Node                  // Оголошення типу "елемент масиву"
{  int data;
   Node *next;
};
int main()
{
  Node* head = NULL;         // Вказівник на початок списку
  Node* c;                   // Поточний елемент списку
  Node* end = NULL;          // Вказівник на останній елемент списку
  do {                       // Створення нового елемента окремо від списку
    c = new struct Node;
    c->next = NULL;
    cin>>c->data;
    // Приєднання до списку
    if (head==NULL)          // Якщо списку ще немає,
    {   head=c;   end=c;     // то новий елемент стає першим і останнім
    }
    else // Якщо список вже є,
    { end->next=c; // то записати його адресу в адресне поле останнього елемента і
      end=c;       // перемістити вказівник end на новий елемент, бо він тепер є останнім
    }
  } while (c->data!=-1000);  // доки не введено число -100
  // Виведення списку на екран
  c=head;                    // Стати на перший елемент
  while (c!=NULL)            // Доки список не закінчився,
  {  cout<<c->data<<" ";     // вивести значення чергового елемента
     c=c->next;              // і перейти до наступного елемента
  }
  cout<<endl;
  return 0;
```

```
}
```

Результати:

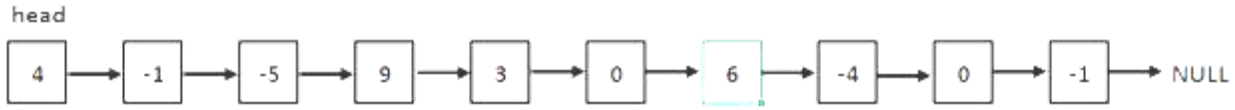
```
5
-2
-1
8
5
34
0
-3
 4
-1000
5 -2  -1  8  5  34  0  -3  4  -1000
```

Приклад 2. Сформувати список з 10 цілих чисел. Обчислити суму додатних елементів.

```
#include<iostream>
using namespace std;
struct Node
{
  int data;
  Node *next;
};
int main()
{
  Node* head = NULL;
  Node* c;
  Node* end = NULL;
  int sum = 0;
  cout << "Ввести 10 цілих чисел:" << endl;
  for(int i=1; i<=10; i++)
  {
    c = new struct Node;
    c->next = NULL;
    cin>>c->data;
    if (head==NULL)
    {
      head=c; end=c;
    }
    else
    {
      end->next=c;
      end=c;
    }
  }
```

```
 cout<<"The list"<<endl;
 c=head;
 while (c!=NULL)
 {  cout<<c->data<<" ";
    c=c->next;
 }
 cout<<endl;
 c=head;                  // Стати на початок списку
 while (c!=NULL)          // Доки не кінець списку
 {
   if (c->data > 0)       // Якщо значення елемента додатне,
      sum+=c->data;       // додати його до суми і
   c=c->next;             // перейти до наступного елемента
 }
 cout<< "Sum of positive elements = " << sum << endl;
 return 0;
}
```

Результати:




```
4
-1
-5
9
3
0
6
-4
0
-1
The list
4 -1 -5 9 3 0 6 -4 0 -1
Sum of positive elements = 22
```

Приклад 3. Ввести 10 цілих чисел до списку. Знайти максимальний елемент.

```
#include <iostream>
using namespace std;
struct Node
{
  int data;
```

```
  Node *next;
};
int main()
{
  Node* head = NULL, *end = NULL, *c;
  Node* max = NULL; // Максимальний елемент
  cout << "Введіть 10 елементів" << endl;
  for (int i = 1; i <= 10; i++)
  { c = new Node;
    c->next = NULL;
    cin >> c->data;
    if (head == NULL)
    {  head = c; end = c;
    }
    else
    {  end->next = c;
       end = c;
    }
  }
  cout << "The list" << endl;
  c = head;
  while (c != NULL)
  {  cout << c->data << " ";
     c = c->next;
  }
  cout << endl;
  c = head;          // Стати на початок списку
  max = head;      // Початкове значення максимуму - перший елемент
  while (c != NULL)
  {  // Якщо значення елемента більше за значення максимуму
     if (c->data > max->data)   max = c;
     c = c->next;
  }
  cout << "Max = " << max->data << endl;
  return 0;
}
```

Результати:

Введіть 10 елементів
5
0
-3
7
2
56
3
8

```
-1
-2
The list
5   0   -3   7   2   56   3   8   -1   -2
Max = 56
```

Приклад 4. Вводити дійсні числа до списку, доки їхня сума не стане більшою за 50. Поміняти місцями перший та останній елементи.

```
#include <iostream>
using namespace std;
struct Node
{
    double data;
    Node *next;
};
int main()
{
  Node* head = NULL, *end, *c;
  double sum=0;
  // Вводити елементи, поки їхня сума менша за 50
  while (sum < 50)
  {
    c = new Node;
    c->next = NULL;
    cin >> c->data;
    sum += c->data;
    if (head == NULL)
    {
      head = c;
      end = c;
    }
    else
    {
      end->next = c;
      end = c;
    }
  }
  cout << "The list before: " << endl;
  c = head;
  while (c != NULL)
  {
    cout << c->data << "  ";
    c = c->next;
  }
  cout << endl;
```

```
// Поміняти перший і останній елементи
  double x = head->data;
  head->data = end->data;
  end->data = x;
  cout << "The list after: " << endl;
  c = head;
  while (c != NULL)
  {
    cout << c->data << " ";
    c = c->next;
  }
  cout << endl;
  return 0;
}
```

Результати:

```
9
-3
12
31
-8
5
8
The list before:
9   -3   12   31   -8   5   8
The list after:
8   -3   12   31   -8   5   9
```

Контрольні запитання для самоконтролю

1) Що таке структура даних «список»?
2) Чи кожен список є зв'язним? Обґрунтуйте відповідь.
3) У чому відмінність першого елемента односпрямованого списку від інших елементів цього ж списку?
4) У чому відмінність останнього елемента односпрямованого списку від інших елементів цього ж списку?
5) Чому при роботі з односпрямованим списком необхідне позиціонування на перший елемент списку?
6) З якою метою в програмах виконується перевірка на порожнину односпрямованого списку?

Лабораторне завдання

Розробити програму зі створення й опрацювання зв’язних лінійних списків (завдання обрати з табл. 1.1, 1.2). Вводити числа до списку, доки не введено число 999 (табл. 1.1) і ввести 10 елементів (табл. 1.2).

Зобразити створені списки у протоколі лабораторної роботи.

Таблиця 1.1

№ вар.	Тип даних	Індивідуальне завдання
1	дійсний	Обчислити суму модулів всіх від'ємних елементів списку
2	цілий	Визначити найбільший з парних додатних елементів
3	дійсний	Обчислити суму квадратів тих чисел, модуль яких більше 2.5
4	цілий	Обчислити процентний вміст додатних, від'ємних і нульових елементів
5	дійсний	Визначити максимальний і мінімальний елементи, обчислити наскільки максимальний елемент є більшим за мінімальний
6	дійсний	Обчислити суму тільки двоцифрових елементів
7	цілий	Обчислити добуток одноцифрових елементів списку
8	цілий	Обчислити кількість додатних, від'ємних і нульових елементів
9	дійсний	Визначити мінімальний із додатних елементів
10	цілий	Обчислити середнє арифметичне елементів, кратних 5
11	дійсний	Обчислити модуль суми всіх від'ємних елементів, суму всіх додатних та різницю між значеннями цих сум
12	дійсний	Обчислити кількість елементів, значення яких менші 20-ти
13	цілий	Обчислити середнє арифметичне елементів, значення яких більше значення останнього елемента списку
14	цілий	Визначити кількість двоцифрових елементів
15	дійсний	Обчислити добуток мінімального і максимального елементів списку
16	цілий	Обчислити добуток непарних елементів списку
17	дійсний	Обчислити суму елементів списку, значення яких належать проміжку [3, 9]
18	цілий	Визначити мінімальний та максимальний елементи списку
19	дійсний	Вивести кількість елементів, значення яких більше за значення першого елемента списку
20	цілий	Вивести на екран елементи, які за модулем менші, ніж перший елемент
21	дійсний	Обчислити кількість елементів, які розташовані до елемента зі значенням 10, і вивести їх на екран
22	цілий	Вивести на екран елементи, які діляться на 2 і не діляться на 5
23	дійсний	Визначити середнє арифметичне елементів, що за модулем не перевищують 12
24	цілий	Вивести на екран двоцифрові елементи й обчислити їхню суму
25	дійсний	Обчислити суму модулів елементів, менших за перший елемент

Таблиця 1.2

№ вар.	Тип даних	Індивідуальне завдання
1	дійсний	Обчислити суму парних елементів, розміщених після мінімального елемента списку
2	цілий	Обчислити кількість елементів, розміщених після першого нульового елемента списку
3	цілий	Замінити всі непарні елементи списку одиницями
4	дійсний	Замінити всі від'ємні елементи на нулі, а додатні – на одиниці
5	дійсний	Обчислити суму від'ємних елементів, розміщених після максимального елемента списку
6	цілий	Замінити всі нульові елементи значенням мінімального елемента
7	дійсний	Визначити найбільший серед від'ємних елементів списку
8	цілий	Створити новий список із залишків від ділення націло елементів списку на 3
9	цілий	Визначити найменший серед парних додатних елементів
10	цілий	Замінити парні по значенню елементи на 0
11	дійсний	Замінити мінімальний і максимальний елементи значенням середнього арифметичного всіх елементів
12	дійсний	Обчислити кількість елементів, що перевищують середнє арифметичне всіх елементів
13	цілий	Замінити всі від'ємні елементи мінімальним
14	дійсний	Обчислити новий список як різницю елементів вихідного списку та їх середнього арифметичного
15	дійсний	Поміняти місцями максимальний елемент з першим
16	цілий	Обчислити факторіал першого елемента списку, значення якого менше 8-ми
17	цілий	Перевірити, чи є список упорядкованим за зростанням
18	цілий	Обчислити суму додатних непарних елементів і замінити парні елементи списку на цю суму
19	дійсний	Поміняти місцями мінімальний елемент з передостаннім
20	дійсний	Визначити кількість максимальних елементів списку
21	цілий	Обчислити суму елементів, розташованих між першим і останнім додатними елементами у списку
22	дійсний	Перевірити, чи є список упорядкованим за спаданням модулів
23	цілий	Обчислити факторіал мінімального за модулем елемента
24	дійсний	Обчислити добуток елементів, розташованих між першим і останнім від'ємними елементами
25	цілий	Визначити найменший серед додатних елементів списку

Лабораторна робота 2

Лінійні списки: вставлення і вилучення елементів

Теоретичні відомості

Вставлення елемента у список

Вставити елемент у список можна:
1) на початок списку (перед першим елементом);
2) після заданого елемента;
3) у кінець списку (після останнього елемента).
Розглянемо всі ці варіанти докладно.

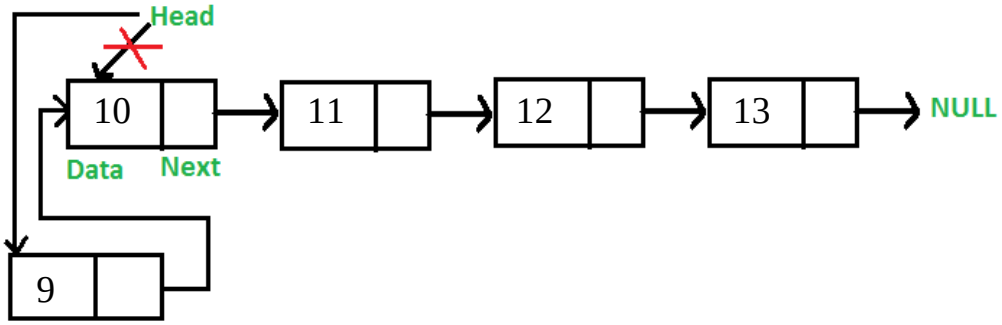
1) Вставлення елемента на початок списку

При такому вставленні елемент, що вставляється, стає першим елементом списку. Наприклад, якщо список був такий:

10 -> 11 -> 12 -> 13

Після долучення елемента зі значенням 9 на початок списку, список стане таким:

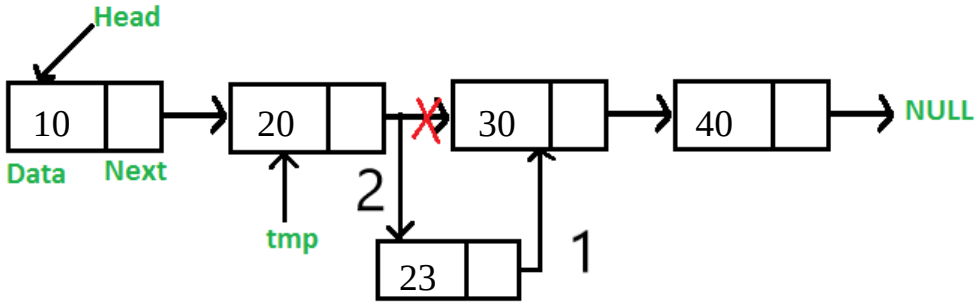
9 -> 10 -> 11 -> 12 -> 13.



Функція вставлення нового елемента на початок списку матиме два аргументи: вказівник на початок списку і числове значення, яке треба вставити. Саме долучення можна виконати в 4 кроки:

```
void Insert_Beg(Node* &head, int new_data)
{
   // 1. Виділити місце в пам'яті для нового елемента
  Node* new_node = new Node;
   // 2. Присвоїти значення  нового елемента
   new_node->data  = new_data;
   // 3. Зв'язати новий елемент з першим
   new_node->next = head;
   // 4. Новий елемент стає першим
   head   = new_node;
}
```

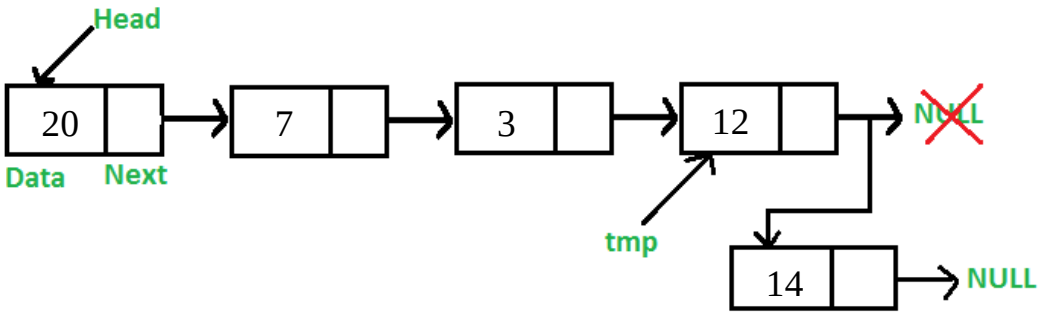
2) Вставлення після заданого елемента списку



Функція такого вставлення нового елемента аргументами матиме: вказівник на елемент списку prev_node, після якого треба вставити новий елемент, і числове значення, яке треба вставити. Саме долучення можна виконати за 5 кроків:

```
void Insert_After(Node* prev_node, int new_data)
{
   if (prev_node == NULL)                  // 1. Перевірити, чи існує заданий елемент
   {
     cout<<"The given previous node cannot be NULL"<<endl;  return;
   }
   Node* new_node = new Node;              // 2. Створити новий елемент
   new_node->data  = new_data;             // 3. Записати у нього дані
   new_node->next = prev_node->next; // 4. Наступним після нового стає prev_node->next
   prev_node->next = new_node;             // 5. Наступним після prev_node стає новий
}
```

3) Вставлення у кінець списку (після останнього елемента)



Функція такого вставлення нового елемента аргументами матиме: вказівники на початок і кінець списку і числове значення, яке треба вставити:

```
void Append(Node* &head, Node* &end, int x)
{
    Node* c = new Node;   // 1. Створити новий елемент
    c->data = x;                // 2. Записати у новий елемент дані
    c->next = NULL; // 3. Новий вузол буде останнім, тому він у нього немає наступного
     // 4. Якщо зв'язаний список порожній, зробити новий вузол заголовком
      if (head == NULL)
      {
```

```
      head = c; end = c;
    }
   else    // 5. Інакше до останнього вузла долучити новий вузол
    {
      end->next = c;
      end = c;
    }
    return;
}
```

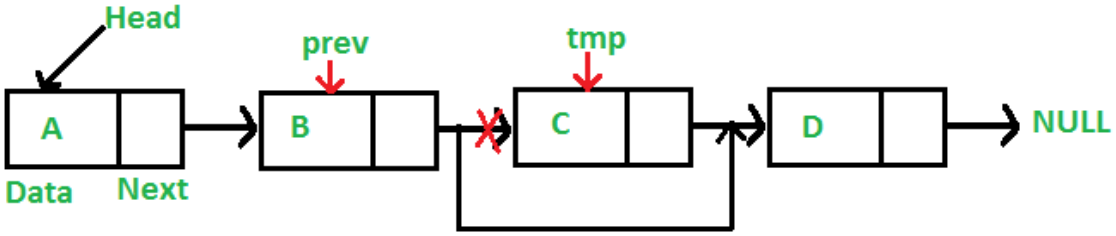
Вилучення елемента зі списку

Для вилучення вузла зі зв'язаного списку треба виконати такі кроки:

1) найти попередній вузол, тобто вузол, пясля якого йде вузол, який треба вилучити;

2) змінити зв'язок наступного і попереднього вузлів;

3) звільніти пам'ять від вузла, який видаляється.



Функція для вилучення першого входження елемента **x** у зв'язаному списку аргументами матиме: вказівники на початок і кінець списку і числове значення, яке треба вилучити:

```
void deleteNode(Node* &head, Node* &end, int x)
{
   // Store head node
   Node* temp = head, *prev = NULL;

   // Якщо видалити треба перший елемент (голову)
   if (temp != NULL && temp->data == x)
   {
      head = temp->next; // Змінити голову на наступний елемент у списку
      delete temp; // і звільнити пам'ять
      return;
   }
  // Пошук елемента, який потрібно видалити   /
  while (temp != NULL && temp->data != x)
  {
     prev = temp; // відстежуємо попередній вузол, оскільки треба змінити 'prev->next'
     temp = temp->next;
  }
```

```
    // Якщо елемент x відсутній у зв'язаному списку
    if (temp == NULL) return;
    // Від'єднати елемент від зв'язаного списку
    prev->next = temp->next;
    // Якщо елемент x є останнім у списку
    if (temp == end) end = prev;
    delete temp; // Звільнити пам’ять
  }
```

Приклади програм

Приклад 1. Створити список з 10 цілих чисел. Вставити 100 на початок списку. Вставити 200 після елементів зі значенням 5. Вставити 300 у кінець списку. Вилучити перший елемент зі значенням 7. Вилучити елемент 100. Вилучити 300. Вставити 400 у кінець списку.

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node *next;
};
void Add_Node(Node* &head, Node* &end, int x)
{
  Node* c = new Node;
  c->next = NULL;
  c->data = x;
  if (head == NULL)
  {  head = c;
     end = c;
  }
  else
  {  end->next = c;
     end = c;
  }
}
void Print_List(Node* head)
{  cout << "The list:" << endl;
   Node* c = head;
   while (c != NULL)
   {   cout << c->data << " ";
       c = c->next;
   }
   cout << endl;
}
```

```
void Free_Memory(Node* head)
{
  Node* c = head;
  while (c != NULL)
  {
    head = head->next;
    delete c;
    c = head;
  }
}
void Insert_Beg(Node* &head, int new_data)
{
  // 1. Виділити місце в пам'яті під новий вузол
  Node* new_node = new Node;

  // 2. Записати у нього дані
  new_node->data = new_data;

  // 3. Зв'язати новий вузол з першим
  new_node->next = head;

  // 4. Перемістити head на новий вузол
  head = new_node;
}
void Insert_After(Node* prev_node, int new_data)
{
  //1. check if the given prev_node is NULL
  if (prev_node == NULL)
  {
    cout << "The given previous node cannot be NULL" << endl;
    return;
  }
  // 2. allocate new node
  Node* new_node = new Node;
  // 3. put in the data
  new_node->data = new_data;

  // 4. Make next of new node as next of prev_node
  new_node->next = prev_node->next;

  // 5. move the next of prev_node as new_node
  prev_node->next = new_node;
}
void Append(Node* &head, Node* &end, int x)
{
  // 1. allocate node
  Node* new_node = new Node;
```

```
  // 2. put in the data
  new_node->data = x;
  // 3. This new node is going to be the last node, so make next of it as NULL
  new_node->next = NULL;
  // * 4. If the Linked List is empty, then make the new node as head
  if (head == NULL)
  {
    head = new_node; end = new_node;
  }
  else
  // 5. Change the next of last node
  {
    end->next = new_node; end = new_node;
  }
  return;
}

void deleteNode(Node* &head, Node* &end, int x)
{
  // Store head node
  Node* temp = head, *prev = NULL;

  // If head node itself holds x (the key to be deleted)
  if (temp != NULL && temp->data == x)
  {
    head = temp->next; // move head to the second element
    delete temp; // free old head
    return;
  }
  // Search for the key to be deleted, keep track of the
  // previous node as we need to change 'prev->next'
  while (temp != NULL && temp->data != x)
  {
    prev = temp;
    temp = temp->next;
  }
  // If key x was not present in linked list
  if (temp == NULL) return;
  // Unlink the node from linked list
  prev->next = temp->next;
  // If key x is the last
  if (temp == end) end = prev;
  delete temp; // Free memory
}
```

```
int main()
{
  Node* head = NULL, *end = NULL;
  int x;
  cout << "Input 10 elements:" << endl;
  for (int i = 1; i <= 10; i++)
  {
    cin >> x;
    Add_Node(head, end, x);
  }
  Print_List(head);
  cout << "\nInsert 100 at the beginning"<< endl;
  Insert_Beg(head, 100);
  Print_List(head);
  cout << "\nInsert 200 after element with value 5" << endl;
  Node* prev_node = head;
  while (prev_node->data != 5)
     prev_node = prev_node->next;
  Insert_After(prev_node, 200);
  Print_List(head);
  cout << "\nInsert 300 after the last element" << endl;
  Append(head, end, 300);
  Print_List(head);
  cout << "\nDelete the first 7" << endl;
  deleteNode(head, end, 7);
  Print_List(head);
  cout << "\nDelete 100" << endl;
  deleteNode(head, end, 100);
  Print_List(head);
  cout << "\nDelete 300" << endl;
  deleteNode(head, end, 300);
  Print_List(head);
  cout << "\nInsert 400 after the end" <<
  endl;
  Append(head, end, 400);
  Print_List(head);
  Free_Memory(head);
  return 0;
}
```

Результати:

```
Input 10 elements:
1 2 3 4 5 6 7 8 9 0
The list:
1 2 3 4 5 6 7 8 9 0

Insert 100 at the beginning
The list:
100 1 2 3 4 5 6 7 8 9 0

Insert 200 after element with value 5
The list:
100 1 2 3 4 5 200 6 7 8 9 0

Insert 300 after the last element
The list:
100 1 2 3 4 5 200 6 7 8 9 0 300

Delete the first 7
The list:
100 1 2 3 4 5 200 6 8 9 0 300

Delete 100
The list:
1 2 3 4 5 200 6 8 9 0 300

Delete 300
The list:
1 2 3 4 5 200 6 8 9 0

Insert 400 after the end
The list:
1 2 3 4 5 200 6 8 9 0 400
Для продолжения нажмите любую клавишу . . .
```

Приклад 2. Створити список з 10 цілих чисел. Вставити елементі зі значенням 1 перед кожним елементом зі значенням 5.

```
#include <iostream>
using namespace std;

struct Node {
  int data;
  Node *next;
};
```

```
void Add_Node(Node* &head, Node* &end, int x)
{
  Node* c = new Node;
  c->next = NULL;
  c->data = x;
  if (head == NULL)
  {
    head = c; end = c;
  }
  else
  {  end->next = c;
    end = c;
  }
}
void Print_List(Node* head)
{
  cout << "The list:" << endl;
  Node* c = head;
  while (c != NULL)
  {
    cout << c->data << " ";
    c = c->next;
  }
  cout << endl;
}
void Free_Memory(Node* head)
{
  Node* c = head;
  while (c != NULL)
  {
    head = head->next;
    delete c;
    c = head;
  }
}
void Insert_Beg(Node* &head, int new_data)
{
  Node* new_node = new Node;
  new_node->data = new_data;
  new_node->next = head;
  head = new_node;
}
```

```
void Insert_After(Node* prev_node, int new_data)
{
  if (prev_node == NULL)
  {   cout << "The given previous node cannot be NULL" << endl;
     return;
  }
  Node* new_node = new Node;
  new_node->data = new_data;
  new_node->next = prev_node->next;
  prev_node->next = new_node;
}
void Example2(Node* &head)
{
  Node* c = head;
  if (head->data == 5)
      Insert_Beg(head, 1);
  while (c->next != NULL)
  {
    if (c->next->data ==5) // If the next element is 5
    {  // Insert 1 after the current element
       Insert_After(c, 1); c = c->next;
    }
    c = c->next;
  }
}
int main()
{
  Node* head = NULL, *end = NULL;
  int x;
  cout << "Input 10 elements:" << endl;
  for (int i = 1; i <= 10; i++)
  {  cin >> x;
     Add_Node(head, end, x);
  }
  Print_List(head);
  cout << "\nInsert 1 before all 5" << endl;
  Example2(head);
  Print_List(head);
  Free_Memory(head);
  return 0;
}
```

Результати:

```
Input 10 elements:
5 5 6 2 3 5 6 7 4 5
The list:
5 5 6 2 3 5 6 7 4 5

Insert 1 before all 5
The list:
1 5 1 5 6 2 3 1 5 6 7 4 1 5
```

Приклад 3. Створити список з 10 цілих чисел. Вилучити зі списку всі елементи, які дорівнюють 5.

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   Node *next;
};
void Add_Node(Node* &head, Node* &end, int x)
{
  Node* c = new Node;
  c->next = NULL;
  c->data = x;
  if (head == NULL)
  {    head = c; end = c;
  }
  else
  {   end->next = c; end = c;
} }
void Print_List(Node* head)
{
  cout << "The list:" << endl;
  Node* c = head;
  while (c != NULL)
  {   cout << c->data << " ";
      c = c->next;
  }
  cout << endl;
}
void Free_Memory(Node* head)
{
  Node* c = head;
  while (c != NULL)
  {
      head = head->next;
      delete c;
      c = head;
  }
}
// Функція видаляє зі списку елемент, на який вказує ptr,
// повертає вказівник на елемент після видаленого
Node* deleteNodePtr(Node* &head, Node* &end, Node* ptr)
{
  if (ptr == NULL) return NULL;
  Node* c = head;
```

```
  //Якщо перший елемент - це ptr, видаляємо його і повертаємо вказівник на head
  if (ptr == head)
  {
     head = head->next;
     delete c; return head;
  }
  //Перебираємо елементи списку. Зупиняємось на елементі ПЕРЕД ptr
  //або коли список скінчиться
  c = head;
  while (c->next != ptr && c != NULL)
     c = c->next;
  //Якщо список закінчився, а елемента ptr не знайшли, то виходимо
  if (c == NULL) return NULL;
  //c - елемент ПЕРЕД тим, що треба вилучити,
  //ptr->next - це елемент ПІСЛЯ того, що треба вилучити
  c->next = ptr->next;  //Зв'язуємо c і ptr->next
  //Якщо ptr був останнім елементом списку, переміщуємо end на попередній (c)
  if (ptr == end) end = c;
  delete ptr;  //Звільнюємо пам'ять від ptr
  return c->next;  //Повертаємо у точку виклику наступний елемент після видаленого
}
int main()
{
  Node* head = NULL, *end = NULL, *c, *ptr;
  int x;
  cout << "Input 10 elements:" << endl;
  for (int i = 1; i <= 10; i++)
  {
     cin >> x;
     Add_Node(head, end, x);
  }
  Print_List(head);
  cout << "\nDelete all 5" << endl;
  //Проходимо списком
  c = head;
  while (c != NULL)
  {
     // Якщо зустріли елемент зі значенням 5
     if (c->data == 5)
     {
        // Видаляємо його, в c повернеться наступний після видаленого
        c = deleteNodePtr(head, end, c);
     }
     else c = c->next; // інакше переходимо на наступний елемент
  }
```

```
  cout << endl;
  Print_List(head);
  Free_Memory(head);
  return 0;
}
```

Результати:

```
Input 10 elements:
5 5 6 2 3 5 6 7 4 5
The list:
5 5 6 2 3 5 6 7 4 5

Insert 1 before all 5
The list:
1 5 1 5 6 2 3 1 5 6 7 4 1 5
```

Контрольні запитання для самоконтролю

1) У чому схожість списків і масивів?
2) Де (у масиві чи у зв'язаному списку) доступ до елемента відбувається швидше?
3) Де (у масиві чи у зв'язаному списку) операції вставки й видалення елемента відбувається швидше?
4) Які переваги зв’язних списків над масивами?
5) З якою метою в програмах виконується видалення односпрямованого списку після закінчення роботи з ним? Як зміниться робота програми, якщо операцію видалення списку не виконувати?

Лабораторне завдання

Розробити програму створення й опрацювання зв’язних списків (табл. 2.1 … 2.2).

Введення чисел до списку:

– базовий рівень – кількість елементів ввести з клавіатури;

– високий рівень складності – вводити елементи відповідно до умови, заданої у варіанті.

Зобразити створений список у протоколі лабораторної роботи. Схематично зобразити вставлення та видалення елементів списку.

Таблиця 2.1

№ вар.	Тип	Умова для введення	Завдання
1	дійсний	Вводити елементи, доки їхня сума не стане менше за їхню кількість	Вилучити зі списку елемент перед кожним елементом зі значенням 55
2	дійсний	Вводити елементи, доки їхня сума не стане більшою за 25	Вилучити зі списку елемент після кожного елемента з від'ємним значенням
3	дійсний	Вводити елементи, доки їхня сума не стане від'ємною	Вставити в список число 1.5 після кожного елемента з додатним значенням
4	цілий	Вводити елементи, доки їхня кількість не стане дільником суми	Вставити число 33 після максимального елемента
5	цілий	Вводити елементи, доки кількість додатних і від’ємних елементів не стане однаковою	Вилучити зі списку перший елемент менший за модулем, ніж 5
6	дійсний	Вводити елементи, доки кількість додатних елементів більше за кількість від’ємних	Продублювати в списку перший додатний елемент (якщо такого немає, залишити список без змін)
7	цілий	Вводити елементи, доки кількість додатних елементів менша за 4	Вилучити зі списку перший парний елемент, що стоїть на непарній позиції
8	цілий	Вводити елементи, доки кількість від’ємних елементів менша за 3	Вставити в список число 66 після кожного елемента з від'ємним значенням
9	цілий	Вводити елементи до першого числа 100	Вилучити зі списку елемент перед кожним елементом із значенням -2
10	цілий	Вводити елементи до першого від’ємного двоцифрового числа	Вилучити зі списку елемент після кожного елемента, рівного 4
11	дійсний	Вводити елементи, доки кількість додатних елементів не стане більшою за 5	Вставити в список число 1.5 після кожного елемента з від'ємним значенням
12	цілий	Вводити елементи, доки кількість від’ємних елементів не стане більшою за 3	Вилучити зі списку всі значення, які менші 5

Продовження табл. 2.1

№ вар.	Тип	Умова для введення	Завдання
13	символи	Вводити елементи до першої великої латинської літери	Вставити символ % після кожної цифри
14	дійсний	Вводити елементи, доки сума додатних елементів не стане більшою за 35	Вставити перший додатний елемент списку після кожного від'ємного числа (якщо такого немає, залишити список без змін)
15	цілий	Вводити елементи, доки сума від'ємних елементів не стане більшою за -10	Вставити в список останній парний елемент після кожного непарного елемента
16	цілий	Вводити елементи, доки сума парних елементів не стане більшою за 20	Вставити в список число 11 після кожного елемента, рівного 9
17	дійсний	Вводити елементи, доки добуток не перевищить 100	Вилучити зі списку елемент перед кожним елементом із значенням в інтервалі від 10 до 20
18	символи	Вводити елементи, доки не зустрінеться символ з кодом 70	Вилучити зі списку елемент після кожного символу &
19	дійсний	Вводити елементи, доки сума від'ємних не стане менша за -23	Вставити в список число 13.5 після першого елемента зі значенням, більшим 2
20	дійсний	Вводити елементи, доки кількість від'ємних не стане більша за 3	Визначити середнє значення елементів списку зі значеннями менше або рівними 15. Вставити це значення перед елементами, які більше 25
21	цілий	Вводити елементи, доки сума непарних елементів не стане більшою за 18	Вилучити зі списку перший елемент, більший числа 4
22	дійсний	Вводити елементи, доки сума елементів не стане від'ємною	Вставити в список перший від'ємний елемент перед кожним числом, рівним 20 (якщо таких немає, залишити список без зміни)
23	цілий	Вводити елементи, доки кількість непарних елементів не стане більшою за 3	Вилучити зі списку кожен елемент, кратний трьом

Закінчення табл. 2.1

№ вар.	Тип	Умова для введення	Завдання
24	цілий	Вводити елементи, доки добуток непарних елементів не стане більшим за 23	Вставити в список число 25 перед кожним елементом з додатним значенням
25	символи	Вводити елементи, доки не введено знак оклику	Вилучити зі списку елемент перед кожним символом ^
26	дійсний	Вводити елементи, доки добуток додатних менше 78	Вилучити зі списку елементи, в яких дійсна частина більше 0,5
27	цілий	Вводити елементи, доки кількість парних елементів менша за 5	Вставити в список число 0 перед кожним числом від 2 до 7. Визначити суму чисел, більших за 7
28	дійсний	Вводити елементи, доки кількість від'ємних менша за 5	Визначити максимальне з елементів списку і вставити його після кожного елемента зі значенням 1
29	цілий	Вводити елементи, доки добуток парних елементів менший за 40	Вилучити зі списку перший елемент кратний 5
30	дійсний	Вводити елементи, доки кількість додатних не перевищить 6	Вставити число 11 перед кожним числом, більшим за середнє значення

Таблиця 2.2

№ вар.	Тип	Умова для введення	Завдання
1	дійсний	Вводити елементи, доки кількість додатних не перевищить 6	Вставити після кожного мінімального елементу максимальний елемент
2	цілий	Вводити елементи, доки добуток парних елементів менший за 40	Вставити середньоарифметичне елементів списку після кожного елемента зі значенням 10
3	дійсний	Вводити елементи, доки кількість від'ємних менша за 5	Вилучити зі списку всі від'ємні елементи

Продовження табл. 2.2

№ вар.	Тип	Умова для введення	Завдання
4	цілий	Вводити елементи, доки кількість парних елементів менша за 5	Вилучити зі списку всі максимальні елементи
5	цілий	Вводити елементи, доки добуток додатних менше 78	Вставити максимальне значення після кожного непарного елемента
6	символи	Вводити елементи, доки не введено знак оклику	Вилучити зі списку всі цифри
7	цілий	Вводити елементи, доки добуток непарних елементів не стане більшим за 23	Вилучити зі списку всі елементи, кратні 3
8	цілий	Вводити елементи, доки кількість непарних елементів не стане більшою за 3	Вилучити зі списку всі парні елементи, які не перевищують 10
9	дійсний	Вводити елементи, доки сума елементів не стане від'ємною	Вставити після кожного від'ємного елемента число 2
10	цілий	Вводити елементи, доки сума непарних елементів не стане більшою за 18	Вставити максимальний елемент після кожного парного додатного елемента
11	дійсний	Вводити елементи, доки кількість від'ємних не стане більша за 3	Вилучити додатні елементи, менші за 12
12	дійсний	Вводити елементи, доки сума від'ємних не стане менша за -23	Вставити введене число перед кожним елементом, більшим за 10
13	символи	Вводити елементи, доки не зустрінеться символ з кодом 70	Вилучити зі списку символ після кожної великої латинської літери
14	дійсний	Вводити елементи, доки добуток не перевищить 100	Вилучити всі елементи, які менші за середньоарифметичне списку
15	цілий	Вводити елементи, доки сума парних елементів не стане більшою за 20	Вилучити всі елементи з абсолютним значенням менше 5
16	цілий	Вводити елементи, доки сума від'ємних елементів не стане більшою за -10	Вилучити всі парні від'ємні елементи
17	дійсний	Вводити елементи, доки сума додатних елементів не стане більшою за 35	Вставити після кожного від'ємного елемента його абсолютне значення

Закінчення табл. 2.2

№ вар.	Тип	Умова для введення	Завдання
18	символи	Вводити елементи до першої великої латинської літери	Вставити перед кожною цифрою дефіс
19	цілий	Вводити елементи, доки кількість від'ємних елементів не стане більшою за 3	Вилучити всі двоцифрові елементи
20	дійсний	Вводити елементи, доки кількість додатних елементів не стане більшою за 5	Вилучити всі елементи, які менші останнього елемента
21	цілий	Вводити елементи до першого від'ємного двоцифрового числа	Вставити суму мінімуму і максимуму перед кожним непарним елементом
22	цілий	Вводити елементи до першого числа 100	Вилучити елементи, які є повними квадратами
23	цілий	Вводити елементи, доки кількість від'ємних елементів менша за 3	Після кожного від'ємного елемента вставити його квадрат
24	цілий	Вводити елементи, доки кількість додатних елементів менша за 4	Вилучити зі списку всі елементи, які дорівнюють модулю першого елемента
25	цілий	Вводити елементи, доки кількість додатних елементів більше за кількість від'ємних	Вставити суму елементів списку перед кожним двоцифровим елементом
26	дійсний	Вводити елементи, доки кількість додатних і від'ємних елементів не стане однаковою	Вставити після кожного від'ємного елемента його модуль
27	цілий	Вводити елементи, доки їхня кількість не стане дільником суми	Вилучити елементи, які зустрічаються в списку більш, ніж 1 раз (залишити по 1 входженню таких елементів)
28	цілий	Вводити елементи, доки їхня сума не стане від'ємною	Вилучити всі елементи, які дорівнюють за модулем останньому елементу
29	дійсний	Вводити елементи, доки їхня сума не стане більшою за 25	Вставити максимальне за модулем значення після кожного від'ємного елемента
30	дійсний	Вводити елементи, доки їхня сума не стане менше за їхню кількість	Вилучити 3 найменші за модулем елементи

Лабораторна робота 3
Двозв'язні лінійні списки

Теоретичні відомості

Для прискорення багатьох операцій доцільно застосовувати переходи між елементами списку в обох напрямках. Це реалізується за допомогою двоспрямованих списків, які є складною динамічною структурою.

Двоспрямований (двозв'язний) список – це структура даних, що складається з послідовності елементів, кожен з яких містить дані і два вказівники на сусідні елементи. При цьому два сусідні елементи повинні містити взаємні посилання один на одного:

Опис найпростішого елемента такого списку виглядає так:

```
struct ім'я_типу {
                    поле даних;
                    адресне поле 1;
                    адресне поле 2;
                };
```

де *поле даних* – це поле будь-якого раніше оголошеного або стандартного типу даних, яке буде безпосередньо зберігати дані;

адресне поле 1 – це вказівник на об'єкт того самого типу, що і визначена структура, в нього записується адреса наступного елемента списку;

адресне поле 2 – це вказівник на об'єкт того самого типу, що і визначена структура, в нього записується адреса попереднього елемента списку.

Наприклад:

```
struct list {
            int data;
            list *next, *prev;
           }
list *headlist ;
```

Тут: elem – поле даних елемента списку; *next, *prev – вказівники на наступний і попередній елементи цієї структури відповідно. Змінна-вказівник headlist задає список як єдиний програмний об'єкт, її значення – вказівник на перший елемент списку.

Основні операції, що виконуються над двозв'язними списками, такі самі, що й для однозв'язного списку.

Оскільки двоспрямований список більш гнучкий, ніж односпрямований, то при долученні елемента у список, потрібно змінювати як вказівник на елемент, за яким відбувається долучення, так і вказівник на елемент, перед яким відбувається долучення. При вилученні елемента зі списку потрібно змінювати як вказівник на сам елемент, що вилучається, так і вказівники на попередній і наступний за ним елементи. Але через те, що елемент двоспрямованого списку має два вказівника, то при виконанні операцій долучення / вилучення елемента треба змінювати більше зв'язків, ніж в односпрямованому списку. Розглянемо основні операції, здійснювані з двоспрямованими списками, а саме:

- створення списку;
- перегляд списку;
- долучення (вставка) елемента у список;
- вилучення (видалення) елемента зі списку;
- пошук елемента у списку;
- перевірка порожнин у списку;
- видалення списку.

Суттєво, що на відміну від односпрямованого списку, тут немає потреби забезпечувати позиціонування будь-якого вказівника саме на перший елемент списку, позаяк завдяки двом вказівникам в елементах можна отримати доступ до будь-якого елемента списку з будь-якого іншого елемента, здійснюючи переходи в прямому або зворотному напрямку. Однак, за правилами хорошого тону програмування вказівник бажано ставити на заголовок списку.

Приклад програми

Створити двозв'язний список з 10-ти елементів. Організувати перегляд елементів списку у прямому та зворотному порядку. Вставити перед кожним від'ємним елементом значення 100.

```
#include <iostream>
using namespace std;
struct Node {
  int data;
  struct Node* next;
  struct Node* prev;
};

/* Given a reference (pointer to pointer) to the head of a
list and an int, inserts a new node on the front of the list. */
void push(struct Node** head_ref, int new_data)
{
  struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
  new_node->data = new_data;
  new_node->next = (*head_ref);
```

```
   new_node->prev = NULL;
   if ((*head_ref) != NULL)
       (*head_ref)->prev = new_node;
   (*head_ref) = new_node;
}

/* Given a node as next_node, insert a new node before the given node */
void insertBefore(struct Node** head_ref, struct Node* next_node, int new_data)
{
/*1. check if the given next_node is NULL */
if (next_node == NULL)
{
  cout <<"the given next node cannot be NULL";
  return;
  }
  /* 2. allocate new node */
  struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
  /* 3. put in the data */
  new_node->data = new_data;
  /* 4. Make prev of new node as prev of next_node */
  new_node->prev = next_node->prev;
  /* 5. Make the prev of next_node as new_node */
  next_node->prev = new_node;
  /* 6. Make next_node as next of new_node */
  new_node->next = next_node;
  /* 7. Change next of new_node's previous node */
  if (new_node->prev != NULL)
      new_node->prev->next = new_node;
  /* 8. If the prev of new_node is NULL, it will be the new head node */
  else
      (*head_ref) = new_node;
}

// This function prints contents of linked list starting from the given node
void printList(struct Node* node)
{
  struct Node* last;
  cout <<"\nTraversal in forward direction \n";
  while (node != NULL)
  {
      cout << node->data<<" ";
      last = node;
      node = node->next;
  }
  cout <<"\nTraversal in reverse direction \n";
```

```
  while (last != NULL)
  {
      cout << last->data<<" ";
      last = last->prev;
  }
}
/* Driver program to test above functions*/
int main()
{
  /* Start with the empty list */
  struct Node* head = NULL, *node;
  int n=10,x;
  cout <<"Input 10 integer numbers: \n";
  for(int i=0;i<n;i++)
  {
      cin>>x;
      push(&head, x);
  }
  cout <<"Created DLL is: ";
  printList(head);
  // Insert 100, before data<0.
  node=head;
  while ( node!= NULL)
  {
      if(node->data<0)
        { insertBefore(&node, node->next, 100); }
      node = node->next;
  }
  cout <<"\nDLL after insert: ";
  printList(head);
  return 0;
}
```

Результати:

```
Input 10 integer numbers:
1 3 -67 0 78 34 -6 -8 45 8
Created DLL is:
Traversal in forward direction
  8  45  -8  -6  34  78  0  -67  3  1
Traversal in reverse direction
1  3  -67  0  78  34  -6  -8  45  8
DLL after insert:
Traversal in forward direction
  8  45  -8  100  -6  100  34  78  0  -67  100  3  1
Traversal in reverse direction
1  3  100  -67  0  78  34  100  -6  100  -8  45  8
```

Контрольні запитання для самоконтролю

1) У чому схожість однозв’язних і двозв’язних списків?
2) У чому відмінність однозв’язних і двозв’язних списків?
3) Яка послідовність дій при долученні елемента у двозв’язний список?
4) Яка послідовність дій при вилученні елемента з двозв’язного списку?
5) Які основні операції виконують з двозв’язними списками?

Лабораторне завдання

Розробити програму створення й опрацювання двозв’язаного лінійного списку (табл. 3.1).

Зобразити створений список у протоколі лабораторної роботи. Схематично зобразити вставлення та видалення елементів списку.

Таблиця 3.1

№ вар.	Індивідуальне завдання
1	Створити лінійний двозв'язний список із цілих чисел. Вилучити зі списку елемент після кожного елемента, рівного 4. Вставити число 0 перед кожним числом 1
2	Створити лінійний двозв'язний список з дійсних чисел. Вставити в список число 1.5 після кожного елемента з від’ємним значенням. Вилучити зі списку всі числа від 2 до 5
3	Створити лінійний двозв'язний список з цілих чисел. Визначити суму елементів списку зі значеннями більше або рівними 15. Вилучити зі списку всі значення, які менші 5
4	Створити лінійний двозв'язний список із символів. Вилучити зі списку перший елемент з ASCII-кодом менше 48. Вставити символ % після кожної цифри
5	Створити лінійний двозв'язний список з дійсних чисел. Вставити перший додатний елемент списку після кожного від’ємного числа (якщо такого немає, залишити список без зміни)
6	Створити лінійний двозв'язний список з цілих чисел. Вставити в список останній парний елемент після кожного непарного елемента
7	Створити лінійний двозв'язний список з цілих чисел. Вставити в список число 11 після кожного елемента, рівного 9
8	Створити лінійний двозв'язний список з дійсних чисел. Вилучити зі списку елемент перед кожним елементом зі значенням в інтервалі від 10 до 20
9	Створити лінійний двозв'язний список із символів. Вилучити зі списку елемент після кожного символу &
10	Створити лінійний двозв'язний список з дійсних чисел. Вставити в список число 13.5 після першого елемента зі значенням більшим 2

Закінчення табл. 3.1

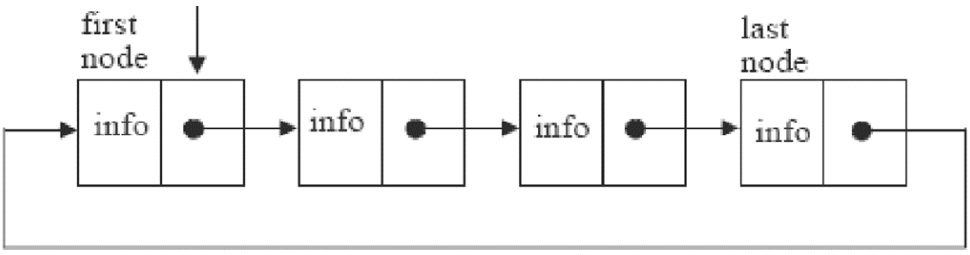
№ вар.	Індивідуальне завдання
11	Створити лінійний двозв'язний список з дійсних чисел. Визначити середнє значення елементів списку зі значеннями менше або рівними 15. Вилучити зі списку елементи, які більше 25
12	Створити лінійний двозв'язний список з цілих чисел. Вилучити зі списку перший елемент, більший числа 4. Вставити в список число 10 перед кожним числом, рівним 15
13	Створити лінійний двозв'язний список з дійсних чисел. Вставити в список перший від'ємний елемент перед кожним числом, рівним 20 (якщо таких немає, залишити список без зміни)
14	Створити лінійний двозв'язний список з цілих чисел. Вилучити зі списку кожний елемент, кратний трьом. Вставити в список число 88 після кожної пари рівних чисел, що розташовані поруч
15	Створити лінійний двозв'язний список з цілих чисел. Вставити в список число 25 перед кожним елементом з додатним значенням. Вилучити зі списку всі від'ємні числа
16	Створити лінійний двозв'язний список із символів. Вилучити зі списку елемент перед кожним символом ^. Визначити кількість символів * у списку
17	Створити лінійний двозв'язний список з дійсних чисел. Вилучити зі списку елементи, у яких дробова частина більше 0,5
18	Створити лінійний двозв'язний список з цілих чисел. Вставити в список число 0 перед кожним числом від 2 до 7. Визначити суму чисел, більших за 17
19	Створити лінійний двозв'язний список з дійсних чисел. Визначити середнє арифметичне від'ємних елементів списку і вставити його перед і після кожного елемента зі значенням 1
20	Створити лінійний двозв'язний список з цілих чисел. Перед кожним двоцифровим числом вставити суму цифр цього числа
21	Створити лінійний двозв'язний список з дійсних чисел. Визначити максимальне з від'ємних елементів списку і вставити його після кожного додатного елемента
22	Створити лінійний двозв'язний список з цілих чисел. Після кожного трицифрового числа вставити добуток його цифр
23	Створити лінійний двозв'язний список із дійсних чисел. Після кожного від'ємного елемента вставити більший з його сусідніх елементів
24	Створити лінійний двозв'язний список з цілих чисел. Вилучити зі списку перший елемент кратний 5. Вставити число 44 перед кожним числом кратним 7
25	Створити лінійний двозв'язний список з дійсних чисел. Обчислити середнє значення елементів списку і вставити число 11 перед кожним числом, більшим за середнє значення

Лабораторна робота 4
Циклічні однозв'язні списки

Теоретичні відомості

У лінійному зв'язному списку після перебору елементів списку, коли ми дістаємось кінця списку, обов'язково потрібен додатковий вказівник на перший елемент списку (head), щоб не втратити доступ до списку і мати можливість нового перегляду його елементів.

Циклічний список дає можливість переглядати елементи списку знов і знов. Циклічний список подібний до лінійного. Відмінність полягає в тому, що останній елемент (end) зв'язаний з першим (head), а у лінійному списку наступним після останнього є NULL.



Приклад програми

Створити циклічний список і вилучити з нього всі максимальні елементи.

```
#include <iostream>
using namespace std;
struct Node {
  int data;
  Node *next;
};
Node* InitList(double x)
{
  Node* c = new Node;
  c->next = c;
  c->data = x;
  return c;
}
Node* AddNodeAfter(Node* &ptr, double x)
{
   Node* c = new Node;
  c->next = c;
  c->data = x;
  if (ptr == NULL) return NULL;
```

```
  else
  { c->next = ptr->next;
    ptr->next = c;
    ptr = c;
    return c->next;
} }
void Print_List(Node* head)
{
  cout << "The list:" << endl;
  Node* c = head;
  do {
    if (c) {
      cout << c->data << " ";
      c = c->next;
    }
  } while (c != head);
  cout << endl;
}
void Free_Memory(Node* head, Node* end)
{
  if (head == NULL) return;
  Node* c = head;
  end->next = NULL;
  while (c != NULL)
  {  head = head->next;
     delete c;
     c = head;
} }
Node* DeleteNode(Node* &ptr, Node* &head, Node* &end)
{
  if (ptr == NULL) return NULL;
  if (ptr == head)
  {  if (head == end)
     {  delete ptr;
        head = end = NULL;
        return NULL;
     }
    else
   {  head = head->next;
      end->next = head;
      delete ptr;
      return head;
   } }
  Node* prev = ptr->next;
  while (prev && prev->next != ptr)
       prev = prev->next;
  if (prev != ptr)
```

```
  { prev->next = ptr->next;
    if (ptr == end)    end = prev;
    delete ptr;
    return prev->next;
  }
  return NULL;
}
int main()
{
  Node* head = NULL, *end = NULL, *c;  double x;
  Node* max = NULL;
  cout << "Input 7 elements:" << endl;
  for (int i = 1; i <= 7; i++)
  {  cin >> x;
     if (head == NULL) head = end = InitList(x);
      else head = AddNodeAfter(end, x);
  }
  Print_List(head);
  cout << end->next->data << endl;
  c = head; max = head;
  do
  {
    if (c->data > max->data)   max = c;
    c = c->next;
  } while (c != head);
  cout << "\nMax: " << max->data << endl;
  Print_List(head);
  cout << end->next->data << endl;
  c = head; double max2 = max->data;
  do{
    if (c->data == max2)   c = DeleteNode(c, head, end);
     else
        if (c) c = c->next;
    }   while (c != NULL && (c != end || (c==end && end->data==max2)));
  if (head != NULL)
  {  Print_List(head); cout << end->next->data << endl;
  }  else cout << "The list is empty" << endl;
  if (c) Free_Memory(head, end);
  return 0;
}
```

```
Input 7 elements:
4
3
4
3
4
3
4
The list:
4 3 4 3 4 3 4
4

Max: 4
The list:
4 3 4 3 4 3 4
4
The list:
3 3 3
3
```

Контрольні запитання для самоконтролю

1) У чому специфіка циклічних списків?
2) У чому схожість однозв’язних і циклічних списків?
3) Яке значення має останній елемент циклічного списку?

Лабораторне завдання

Розробити програму створення й опрацювання циклічного списку (табл. 4.1).

Зобразити створений список у протоколі лабораторної роботи.

Таблиця 4.1

№ вар.	Індивідуальне завдання
1	Створити циклічний список дійсних чисел. Знайти кількість мінімальних елементів циклічного списку і вилучити всі мінімальні елементи
2	Створити циклічний список дійсних чисел. Визначити, чи є циклічний список впорядкованим за зростанням. Вилучити елементи, які порушують сортування
3	Створити циклічний список цілих чисел. Продублювати в ньому всі парні числа
4	Створити циклічний список дійсних чисел. Перетворити його, додавши до додатних чисел значення максимального елемента. Вилучити останній максимальний елемент
5	Створити циклічний список цілих чисел. Продублювати в ньому всі додатні числа
6	Створити циклічний список дійсних чисел (і додатних, і від'ємних). Визначити, чи є в списку елементи, що перевершують суму всіх елементів списку. Вилучити ці елементи
7	Створити циклічний список дійсних чисел. Визначити, чи утворюють елементи списку геометричну прогресію. Якщо так, то вставити після кожного елемента списку новий елемент зі значенням коефіцієнта прогресії
8	Створити циклічний список цілих чисел. Вилучити зі списку всі мінімальні елементи
9	Створити циклічний список дійсних чисел. Вставити після кожного від'ємного числа новий елемент, рівний його модулю
10	Створити циклічний список дійсних чисел. Вилучити зі списку перший і останній мінімальний елементи
11	Створити циклічний список дійсних чисел. Вставити після кожного максимального елемента новий елемент зі значенням 0
12	Створити циклічний список дійсних чисел. Вставити перед кожним додатним елементом новий елемент, який дорівнює сумі всіх елементів
13	Створити циклічний список дійсних чисел. Вилучити зі списку останній максимальний елемент

Закінчення табл. 4.1

№ вар.	Індивідуальне завдання
14	Створити циклічний список дійсних чисел. Вставити перед першим мінімальним елементом новий елемент, який дорівнює середньому арифметичному елементів списку
15	Створити циклічний список цілих чисел. З кожної пари сусідніх елементів вилучити менший за модулем
16	Створити циклічний список цілих чисел. Вилучити всі елементи, які розташовані між від'ємними числами
17	Створити циклічний список дійсних чисел. Вилучити зі списку всі елементи, значення яких менше середнього арифметичного
18	Створити циклічний список цілих чисел. Вставити після першого числа, кратного 3, новий елемент, який дорівнює сумі всіх додатних елементів списку
19	Створити циклічний список із чисел. Вилучити зі списку всі елементи, які більше попередніх елементів
20	Створити циклічний список дійсних чисел. Вставити після кожного від'ємного елемента його квадрат
21	Створити циклічний список цілих чисел. Вилучити зі списку всі трицифрові парні елементи
22	Створити циклічний список цілих чисел. Вставити перед кожним двоцифровим елементом його останню цифру
23	Створити циклічний список дійсних чисел. Вилучити зі списку всі елементи зі значенням з проміжку (-5, -1)
24	Створити циклічний список цілих чисел. Вставити між кожною парою елементів новий елемент зі значенням суми цих двох елементів
25	Створити циклічний список дійсних чисел. Вилучити зі списку елементи, менші за значенням, ніж середнє арифметичне

Самостійна робота 1

Циклічні двозв’язні списки

Індивідуальні завдання для самостійної роботи

1. Створити циклічний двозв'язний список з дійсних чисел. Визначити суму елементів списку зі значеннями не більше 18. Видалити зі списку елемент перед кожним елементом зі значенням 10. Створити лінійний однозв'язний список з додатних елементів вихідного списку.

2. Створити циклічний двозв'язний список з дійсних чисел. Визначити кількість додатних елементів списку. Видалити зі списку елемент після кожного елемента зі значенням 25. Створити лінійний двозв'язний список з від’ємних елементів вихідного списку.

3. Створити циклічний двозв'язний список з дійсних чисел. Обчислити середнє арифметичне додатних елементів списку. Видалити зі списку елемент після кожного від’ємного елемента. Створити циклічний однозв'язний список з елементів вихідного списку, що не перевищують 10.

4. Створити циклічний двозв'язний список з цілих чисел. Визначити максимальний елемент списку. Видалити зі списку перший парний елемент. Створити лінійний однозв'язний список з непарних елементів вихідного списку.

5. Створити циклічний двозв'язний список з дійсних чисел. Визначити мінімальний елемент списку. Вставити в список числа 1000 після кожного елемента з від'ємним значенням. Створити лінійний двозв'язний список з елементів вихідного списку, більших за 5.

6. Створити циклічний двозв'язний список з дійсних чисел. Визначити суму модулів елементів списку. Визначити суму елементів списку зі значеннями більше або рівними 15. Створити циклічний однозв'язний список із від’ємних елементів вихідного списку, не рівних 10,5.

7. Створити циклічний двозв'язний список з цілих чисел. Визначити кількість тризначних елементів списку. Видалити зі списку останній елемент зі значенням меншим 15. Створити лінійний однозв'язний список з парних елементів вихідного списку.

8. Створити циклічний двозв'язний список з цілих чисел. Визначити добуток двозначних елементів списку. Вставити в список число 600 після кожного елемента з парним значенням. Створити лінійний двозв'язний список з елементів вихідного списку, що належать проміжку [-2,8).

9. Створити циклічний двозв'язний список з дійсних чисел. Визначити добуток додатних елементів списку, що перевищують середнє арифметичне. Вставити в список число 11 перед кожним додатним елементом. Створити циклічний однозв'язний список з елементів вихідного списку, які більше попереднього елемента.

10. Створити циклічний двозв'язний список із дійсних чисел. Визначити середнє арифметичне елементів списку, по модулю менших 20. Видалити зі списку елемент перед першим елементом зі значенням 10. Створити лінійний однозв'язний список з елементів вихідного списку, які перевищують наступний елемент.

11. Створити циклічний двозв'язний список із дійсних чисел. Визначити середнє арифметичне додатних елементів списку. Видалити зі списку елемент після кожного елемента зі значенням 25. Створити лінійний двозв'язний список з елементів вихідного списку, великих, ніж середнє арифметичне.

12. Створити циклічний двозв'язний список із дійсних чисел. Визначити добуток від'ємних елементів списку. Видалити зі списку елемент після першого додатного елементу. Створити циклічний однозв'язний список з елементів вихідного списку, менших, ніж середнє арифметичне.

13. Створити циклічний двозв'язний список із цілих чисел. Визначити суму однозначних елементів списку. Видалити зі списку перший елемент зі значенням -10. Створити лінійний однозв'язний список з елементів вихідного списку, кратних 5.

14. Створити циклічний двозв'язний список із дійсних чисел. Визначити кількість елементів списку, не рівних ні мінімуму, ні максимуму. Вставити в список число 55 після кожного елемента з від'ємним значенням. Створити лінійний двозв'язний список з елементів вихідного списку, які більше своїх сусідів.

15. Створити циклічний двозв'язний список із дійсних чисел. Визначити добуток модулів максимального і мінімального елементів списку. Визначити суму елементів списку зі значеннями більше або рівними 100. Створити циклічний однозв'язний список із елементів вихідного списку, які менше своїх сусідів.

16. Створити циклічний двозв'язний список із цілих чисел. Визначити різницю максимального і мінімального елементів списку. Видалити зі списку останній елемент зі значенням меншим 5. Створити лінійний однозв'язний список із двозначних елементів вихідного списку.

17. Створити циклічний двозв'язний список із цілих чисел. Визначити середнє арифметичне від'ємних елементів списку. Вставити в список число 25 після кожного елементу з парним значенням. Створити лінійний двозв'язний список із однозначних елементів вихідного списку.

18. Створити циклічний двозв'язний список із дійсних чисел. Визначити кількість мінімальних елементів у списку. Вставити в список мінімальний елемент перед кожним елементом із від'ємним значенням. Створити лінійний однозв'язний список із елементів вихідного списку, які понад вдвічі перевищують мінімум.

19. Створити циклічний двозв'язний список із цілих чисел. Визначити суму однозначних елементів списку. Видалити зі списку елемент перед кожним двозначним елементом. Створити лінійний двозв'язний список з від'ємних елементів вихідного списку.

20. Створити циклічний двозв'язний список із дійсних чисел. Визначити добуток від'ємних елементів списку. Видалити зі списку елемент після кожного

максимального елемента. Створити лінійний двозв'язний список з елементів вихідного списку, розташованих до першого додатного елемента.

21. Створити циклічний двозв'язний список із цілих чисел. Обчислити середнє арифметичне елементів списку, кратних 3. Видалити зі списку двозначні елементи. Створити циклічний однозв'язний список з від'ємних елементів вихідного списку.

22. Створити циклічний двозв'язний список із цілих чисел. Визначити максимальний від'ємний елемент списку. Видалити зі списку перший трицифровий елемент. Створити лінійний однозв'язний список з парних одноцифрових елементів вихідного списку.

23. Створити циклічний двозв'язний список із дійсних чисел. Визначити елемент списку з найбільшою сумою сусідніх елементів. Вставити в список цей елемент після кожного нульового елемента Створити лінійний двозв'язний список з елементів вихідного списку, абсолютне значення яких перевищує 123.

24. Створити циклічний двозв'язний список із дійсних чисел. Визначити суму квадратів від'ємних елементів списку. Визначити максимальний елемент списку зі значеннями більше або рівними 10. Створити циклічний однозв'язний список із елементів вихідного списку, які більше своїх сусідів.

25. Створити циклічний двозв'язний список із цілих чисел. Визначити середньоарифметичне двоцифрових від'ємних елементів списку. Видалити зі списку останній парний елемент. Створити лінійний однозв'язний список із чисел, які не перевищують перший елемент списку.

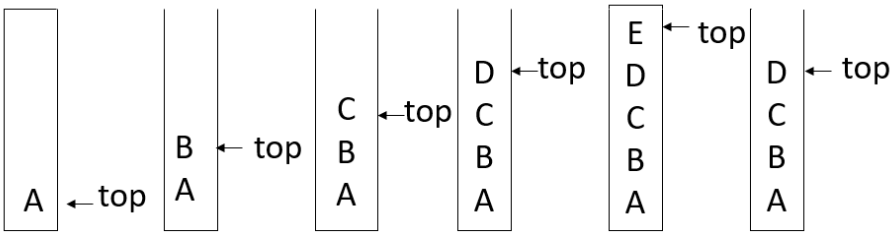
Лабораторна робота 5

Черга і стек

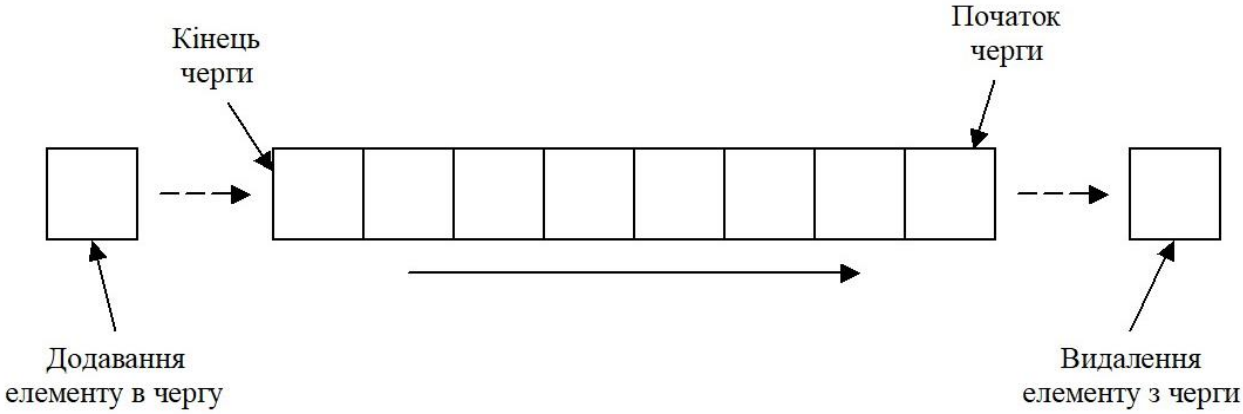
Теоретичні відомості

Стек (stack) і черга (queue) обидва є непримітивними структурами даних. Основними відмінностями між стеком і чергою є те, що стек використовує метод LIFO (last in first out, останнім прийшов – першим пішов) для доступу та долучення елементів даних, тоді як черга використовує метод FIFO (first in first out, першим прийшов – першим вийшов) для доступу та долучення елементів даних.

Стек має тільки один кінець, відкритий для вставлення та «виштовхування» (витягування) елементів даних.



Черга має обидва кінці відкритими для доступу до елементів даних.



Стек і черга є структурами даних, що використовуються для зберігання елементів даних і фактично засновані на деякому еквіваленті реального світу. Наприклад, стек – це стек компакт-дисків, де ви можете витягти та помістити компакт-диск у верхню частину стека компакт-дисків. Аналогічно, чергою є черга, наприклад, за театральними квитками, де особа, що стоїть на першою, отримає квиток першою і покине чергу таж першою.

Приклад програми. Створити стек з парних цілих чисел від 10 до 40. Всі числа з стека витягати по черзі і ті, що кратні 4, записувати в чергу. Вивести вміст сформованої черги. Видалити перший елемент черги та знову переглянути її. Впевнитись, що стек після зняття з нього усіх елементів порожній.

```
#include <iostream>
using namespace std;
struct StackItem //структура - елемент стека
{ int data;
  StackItem* next = nullptr; // вказівник на наступний елемент
};
struct queue // структура - елемент черги
{ int data;
  queue *next;
};
//долучити новий елемент на вершину стека
StackItem* pushSTACK(StackItem* head, int x)
{
  StackItem* new_top = new StackItem;
  new_top->data = x;
  new_top->next = head;
  head = new_top;
  return head;
}
//Долучити новий елемент наприкінець черги
void pushQUEUE(queue *&head, queue *&tail, int x)
{
  queue *element=new queue;
  element->data=x;
  element->next=NULL;
  if(head==NULL)
  head=element;
  else
  tail->next=element;
  tail=element;
}
  void popQUEUE(queue* &head)
{
  head=head->next;
}
StackItem* popSTACK(StackItem* head) // Видалити вершину стека
{
  if (head == nullptr) return 0;
  StackItem* old_top = head;// або auto
  head = old_top->next; //тепер вершина - наступний елемент
  delete old_top;
  return head;
}
```

```
void printSTACK(StackItem* head) // Вивести весь стек
{
  std::cout <<"Стек:\n";
  StackItem* element = head;// або тип auto
  while (element != nullptr) // nullptr (або NULL) - кінець стека
  {
      std::cout << element->data << ' ';
      element = element->next; //перехід до наступного елемента
  }
  std::cout << '\n';
}
void printQUEUE(queue *head) // Вивести всю чергу
{
  queue *element=new queue;
  element=head;
  if (head==NULL) std::cout << "Черга порожня";
  else
  {
      std::cout <<"Черга:\n";
      while (element != nullptr)
      {
          std::cout << element->data << ' ';
          element = element->next;
      }
      std::cout << '\n';
  }
}
int main()
{
  StackItem* top = nullptr;
  // Долучити в стек парні цілі числа від 10 до 40.
  for (int i=10; i<=40; i+=2) top = pushSTACK(top, i);
  printSTACK(top);
  // Всі числа зі стека витягати по черзі і ті, що кратні 4, записувати в чергу.
  queue *head= NULL,*end=NULL;
  StackItem* element = top; // або тип auto
  while (element != nullptr)
  {
    if (element->data % 4 == 0)
       pushQUEUE(head, end, element->data);
    top = popSTACK(top);  // видалити елемент зі стека
    element = top; // перехід до наступного елемента
  }
  std::cout << "Порожній стек після зняття усіх елементів\n";
```

```
  printSTACK(top);
  printQUEUE(head); // Вивести вміст сформованої черги
  std::cout << "Видалити перший елемент черги\n";
  popQUEUE(head); // Видалити перший елемент черги
  printQUEUE(head);
  return 0;
}
```

Результати:

```
Стек:
40 38 36 34 32 30 28 26 24 22 20 18 16 14 12 10
Порожній стек після зняття усіх елементів
Стек:

Черга:
40 36 32 28 24 20 16 12
Видалити перший елемент черги
Черга:
36 32 28 24 20 16 12
```

Контрольні запитання для самоконтролю

1) Структура даних, робота з елементами якої організована за принципом FIFO (перший прийшов - перший пішов), це ... ?
2) Структура даних, робота з елементами якої організована за принципом FILO (першим прийшов - останнім пішов), це ... ?
3) У чому особливості стека?
4) Яке правило вибірки елемента зі стека?
5) Чи можна створити стек (як динамічну структуру даних) на основі лінійного списку?
6) У чому особливості черги?
7) Чи можна чергу (як динамічну структуру даних) створити на основі лінійного двоспрямованого списку?
8) Який тип структур даних використовують у програмах для реалізації буфера, в який можна покласти елемент для подальшої обробки, зберігаючи такий само порядок доступу? Наприклад, якщо база даних підтримує тільки одне з'єднання, можна використати таку структуру даних потоків, які будуть чекати на доступ до БД.

Лабораторне завдання

Розробити програму створення й опрацювання черги і стека (табл. 5.1).

Зобразити створений список у протоколі лабораторної роботи. Схематично зобразити вставлення та вилучення елементів у чергу і стек.

№ вар.	Індивідуальне завдання
1	Створити стек з цілих чисел від 1 до 45. Числа зі стека витягти по одному і вставити в чергу. Числа з черги витягти по одному і парні вивести на екран
2	Створити стек з цілих парних чисел від 4 до 44. Всі числа зі стека витягувати по одному і ті, що більше 10, записувати в чергу. Числа з черги виймати по одному і виводити на екран
3	Створити стек з цілих непарних чисел від 45 до 99. Всі числа з черги витягувати по одному і ті, що менше 77, вставляти в стек, інші виводити на екран. Числа зі стека виймати по одному і виводити на екран
4	Створити стек з цілих непарних чисел від 3 до 55. Всі числа з черги витягувати по одному і записувати в стек. Числа зі стека знімати по одному і виводити на екран ті з них, що кратні 3
5	Ввести з клавіатури число X. Записати в стек парні числа з діапазону від 1 до X, а в чергу непарні з цього діапазону. Витягти 10 чисел зі стека і вивести на екран (якщо чисел у стеку менше 10, вивести повідомлення). Числа з черги виймати по одному і виводити на екран
6	Ввести з клавіатури число X. Записати в стек кратні трьом числа з діапазону від 1 до X, а в чергу – парні з цього діапазону. Всі числа зі стека витягувати по одному і записувати в чергу. Потім числа з черги виймати по одному і парні виводити на екран
7	Ввести з клавіатури число X. Записати в стек парні цілі числа від 4 до 4*X. Всі числа зі стека витягувати по одному і ті, що менше 50, записувати в чергу. Числа з черги виймати по одному і виводити на екран
8	Записати в чергу непарні цілі числа від -3 до 3. Всі числа з черги витягувати по одному і від'ємні записувати в стек, додатні виводити на екран. Числа зі стека виймати по одному і виводити на екран
9	Записати в чергу цілі числа від 1 до 30. Всі числа з черги витягувати по одному і парні записувати в стек. Числа зі стека виймати по одному і більші 10 виводити на екран, а решту записувати в чергу. Числа з черги виймати по одному і виводити на екран
10	Ввести з клавіатури число X. Записати в стек непарні числа з діапазону від 1 до X. Витягти 10 чисел зі стека і вивести на екран (якщо чисел у стеку менше 10, вивести повідомлення), інші витягувати і записувати по одному в чергу. Числа з черги виймати по одному і виводити на екран
11	Записати в чергу непарні цілі числа від 3 до 55. Числа з черги витягувати і ті, що кратні 10, записувати в стек, а ті, що діляться на 5, але не на 10, виводити на екран. Числа зі стека виймати по одному і виводити на екран
12	Ввести з клавіатури число X. Записати в стек парні числа з діапазону від 0 до 3*X, а в чергу – непарні з цього діапазону. Всі числа з черги витягувати по одному і дописувати в стек. Потім числа зі стека виймати по одному і виводити на екран

Закінчення табл. 5.1

№ вар.	Індивідуальне завдання
13	Ввести з клавіатури число X. Записати в стек парні цілі числа від 2 до X. Всі числа зі стека витягувати по одному і записувати в чергу. Три перші числа з черги зняти по одному і вивести на екран
14	Ввести з клавіатури число X. Записати в стек парні цілі числа від 2 до 3*X+1. Вивести на екран вміст стека. Витягти 10 чисел зі стека і записати їх в чергу (якщо чисел у стеку менше 10, вивести повідомлення). Числа з черги виймати по одному і виводити на екран
15	Ввести з клавіатури число X. Записати в стек кратні 3 числа з діапазону від 1 до 2*X+7. Числа зі стека виймати по одному і записувати в чергу. Числа з черги виймати і виводити на екран ті з них, які менше 12
16	Ввести з клавіатури число X. Записати в стек парні цілі числа від 4 до 2*X+2. Всі числа зі стека витягувати по одному і записувати в чергу. Числа з черги виймати по одному і виводити на екран
17	Додати в чергу парні цілі числа від 42 до 88. Всі числа з черги витягувати по одному і ті, що більше 55, записувати в стек, а інші виводити на екран. Числа зі стека виймати по одному і виводити на екран
18	Записати в чергу парні цілі числа від 32 до 52. Всі числа з черги витягувати по одному і записувати в стек. Числа зі стека виймати по одному і виводити на екран ті з них, що кратні 7
19	Ввести з клавіатури число X. Записати в стек кратні 3 числа з діапазону від -X до X, а в чергу – непарні з цього діапазону. Всі числа зі стека витягувати по одному і дописати в чергу. Потім числа з черги виймати по одному і парні виводити на екран
20	Записати в стек цілі числа від -5 до 20. Додатні числа зі стека перенести в чергу. Числа з черги виймати по одному і парні виводити на екран
21	Ввести з клавіатури число X. Записати в стек кратні 3 числа з діапазону від 1 до X, інші з цього діапазону – в чергу. Числа зі стека виймати і виводити на екран. Числа з черги виймати і виводити на екран числа, більші за 10
22	Записати в стек цілі числа від -10 до 50. Числа зі стека витягувати і від'ємні записувати в чергу. Числа з черги виймати по одному і виводити
23	Записати в чергу цілі числа від 50 до 150. Всі числа з черги витягувати по одному і двозначні записувати в стек, інші виводити на екран. Числа з стека виймати по одному і виводити на екран
24	Записати в чергу цілі числа від -20 до 25. Всі числа з черги витягувати по одному і записувати в стек. Числа з стека виймати по одному і виводити на екран ті числа, що кратні 4
25	Записати в стек цілі числа з діапазону від 10 до 65, перенести в чергу непарні числа. Витягти 10 чисел з черги і вивести на екран (якщо чисел в черзі менше 10, вивести повідомлення)

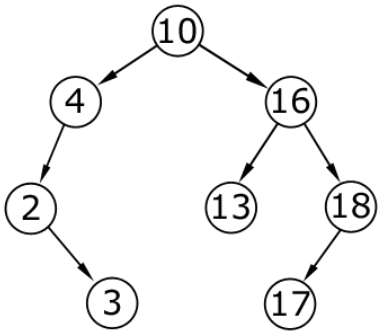
Лабораторна робота 6

Бінарні дерева. Алгоритми пошуку на деревах

Теоретичні відомості

Двійкове (бінарне) дерево – це ієрархічна структура, в якій кожний вузол (елемент) має рівно два зв'язки (вказівники) з можливими нащадками (дітьми). При цьому жодний, один або обидва з них можуть дорівнювати NULL, якщо дітей немає. Існує лише один шлях, який з'єднує будь-які два вузли дерева.

Корінь (початок дерева) – це перший вузол у дереві. Отже, бінарне дерево складається із кореня, лівого піддерева та правого піддерева, причому обидва піддерева є також бінарними деревами. В дереві кожний вузол, окрім кореня, має рівно одного батька (корінь не має батьків).



Кожне посилання від кореня є нащадком (дитиною). Вузол без дітей називають листком.

Однією з причин використання дерев може бути потреба зберігати інформацію, яка природним чином утворює ієрархію, наприклад, файлова система на комп'ютері. Бінарні дерева широко використовують для пошуку даних у спеціально побудованих деревах (базах даних), для оптимізації сортування даних, при обчисленні арифметичних виразів, при кодування методом Хаффмана тощо.

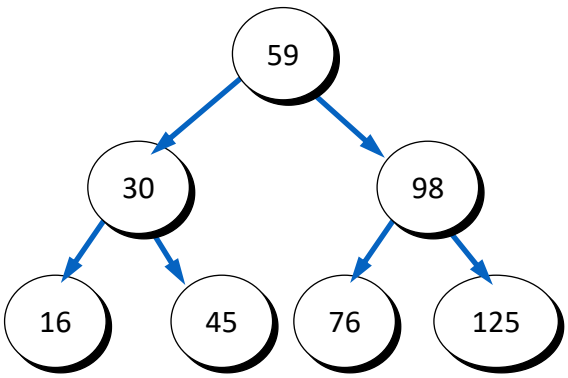
Операції з бінарними деревами:

- пошук вузла;
- вставка нового вузла;
- видалення вузла;
- перегляд дерева (preorder, inorder, postorder).

У **бінарних деревах пошуку** більші елементи розташовуються праворуч, а менші елементи розташовуються ліворуч.

Порядок пошуку елемента (ключа) зі значенням x:

- якщо дерево порожнє, ключ не знайдено;
- якщо ключ вузла дорівнює x, то «стоп».
- якщо ключ вузла менше x, то шукати x у лівому піддереві;
- якщо ключ вузла більше x, то шукати x у правому піддереві.



Структура вузла:

```
struct Node {
  int  data;                  // дані інформаційного поля
  Node *left, *right;         // вказівники на ліву і праву гілку (піддерево)
};
typedef Node *PNode;
```

Приклади програм

Приклад 1. Сформувати бінарне дерево з 7 цілих чисел. Переглянути це дерево у вигляді дерева та у вигляді набору даних (обхід ЛКП). Найти мінімальний та максимальний вузли дерева. Обчислити середнє арифметичне. Найти введений елемент у дереві.

Розв'язок. Нумерація «зліва-направо» дає нам основу для створення різновиду дерев, долучення до яких відбувається не по логічному номеру, а з урахуванням упорядкування значень. Таке дерево називається деревом двійкового (бінарного) пошуку або просто бінарним деревом. Основна властивість, рекурсивно дотримуване для всіх піддерев, виглядає так: у лівому піддереві у кожній вершині є значення, менші ніж у корні, а в правому – більші.

```
#include <iostream>
using namespace std;
struct Node
{  int x;
   Node *l,*r;
};
void show(Node *&Tree) // обхід ЛКП («лівий – корінь – правий»)
{
  if (Tree != NULL) // Доки не зустрінеться порожній вузол
  {  show(Tree->l);
     cout << Tree->x << '\t';
     show(Tree->r);
  }
}
void PrintTree(Node* leaf, int &n)
{
  if (leaf != NULL)
  {  n++;
     PrintTree(leaf->l, n);
     for (int i = 1; i <= n; i++) cout << " ";
     cout << leaf->x<<endl;
     PrintTree(leaf->r, n);
     n--;
} }
```

```
void add_node(int x,Node *&MyTree)
{
 if (NULL == MyTree)
 {  MyTree = new Node;
    MyTree->x = x;
    MyTree->l = MyTree->r = NULL;
 }
 if (x < MyTree->x)
 {   if (MyTree->l != NULL) add_node(x, MyTree->l);
     else
     {
       MyTree->l = new Node;
       MyTree->l->l = MyTree->l->r = NULL;
       MyTree->l->x = x;
 }   }
 if (x > MyTree->x)
 {  if (MyTree->r != NULL) add_node(x, MyTree->r);
    else
    {   MyTree->r = new Node;
        MyTree->r->l = MyTree->r->r = NULL;
        MyTree->r->x = x;
    }
} }
Node *search(Node *&Tree, int &key) // 1 спосіб пошуку
{
  if (Tree==NULL) return NULL;
  if (key==Tree->x) return Tree;
  else
     if (key<Tree->x) return search(Tree->l,key);
     else
         return search(Tree->r,key);
}
Node *search2(Node *tree, int key)  // 2 спосіб пошуку
{
  while (tree!=NULL)
  {  if (key==tree->x) return tree;
     else
        if (key<tree->x)
            tree=tree->l;
        else
           tree=tree->r;
   }
   return NULL;
}
```

```
double average(Node *tree)
{
  double s=0; int kol=0;
  Node *cur=tree;
  while (cur != NULL)
  {  s+=cur->x;
     kol++;
     cur=cur->r;
  }
  cur=tree->l;
  while (cur != NULL)
  {
     s+=cur->x;
     kol++;
     cur=cur->l;
  }
  return s/kol;
}
int max(Node *tree) // 1 спосіб
{
  int rez=tree->x;
  while (tree != NULL)
  {
     rez=tree->x;
     tree=tree->r;
  }
  return rez;
}
int min(Node *tree) // 1 спосіб
{
  int rez=tree->x;
  while (tree != NULL)
  {
     rez=tree->x;
     tree=tree->l;
  }
  return rez;
}
Node* minimum(Node* x) // 2 спосіб
{
  if (x->l == NULL)
     return x;
  return minimum(x->l);
}
```

```
Node* maximum(Node* x) // 2 спосіб
{
  if (x->r == NULL)
     return x;
  return maximum(x->r);
}
void del(Node *&Tree) // очищення пам'яті
{
  if (Tree != NULL)
  {
    del(Tree->l);  // Рекурсивна функція проходу по лівому піддереву
    del(Tree->r);  // Рекурсивна функція для проходу по правому піддереву
    delete Tree;   // Видалити елемент дерева
    Tree = NULL;
  }
}

int main()
{
  Node *Tree = NULL; /
  int x,kol=7;
  for (int i=1; i<=kol; i++)
  {
    cout<<"Input value: ";   cin>>x;
    add_node(x,Tree);
  }
  cout<<"Binary tree from 7 value: \n";
  PrintTree(Tree, kol);
  cout<<"\nBinary tree from 7 value: \n";
  show(Tree);
  cout<<"\nsr ="<<average(Tree)<<endl;
  cout<<"max ="<<max(Tree)<<"="<<maximum(Tree)->x<<endl;
  cout<<"min ="<<min(Tree)<<"="<<minimum(Tree)->x<<endl;
  cout << "Input value for search: ";   cin>>x;
  Node *c=search(Tree, x);
  if(c) cout<<"Your value founded = "<<c->x<<endl;
  else cout<<"Your value not founded! \n";
  c=search2(Tree, x);
  if(c)
     cout<<"Your value founded = "<<c->x<<endl;
  else
     cout<<"Your value not founded! \n";
  del(Tree);
}
```

Результати:

```
Input value: 8
Input value: 3
Input value: 11
Input value: 0
Input value: 4
Input value: 23
Input value: 6
Binary tree from 7 value:
                                        0
                                    3
                                        4
                                            6
                                8
                                    11
                                        23

Binary tree from 7 value:
0   3   4   6   8   11  23
sr =9
max =23=23
min =0=0
Input value for search: 4
Your value founded = 4
Your value founded = 4
```

Приклад 2. Сформувати бінарне дерево з 15 цілих випадкових чисел. Переглянути це дерево у вигляді дерева та у вигляді набору даних (обхід ЛКП). Вивести всі листочки дерева. Найти елемент дерева по значенню, яке вводитиме користувач. За наявності такого елемента у дереві видалити його та знову переглянути дерево після видалення.

```
#include <iostream>
#include <time.h>
using namespace std;
struct Node // Елемент дерева
{
  int x;
  Node *l,*r;  // Вказівники на гілки (елементи)
};
// обхід ЛКП («лівий – корінь – правий»)
void show(Node *&Tree) // Рекурсивна функція обходу дерева
{
 if (Tree != NULL) // Допоки не зустрінеться порожній елемент
```

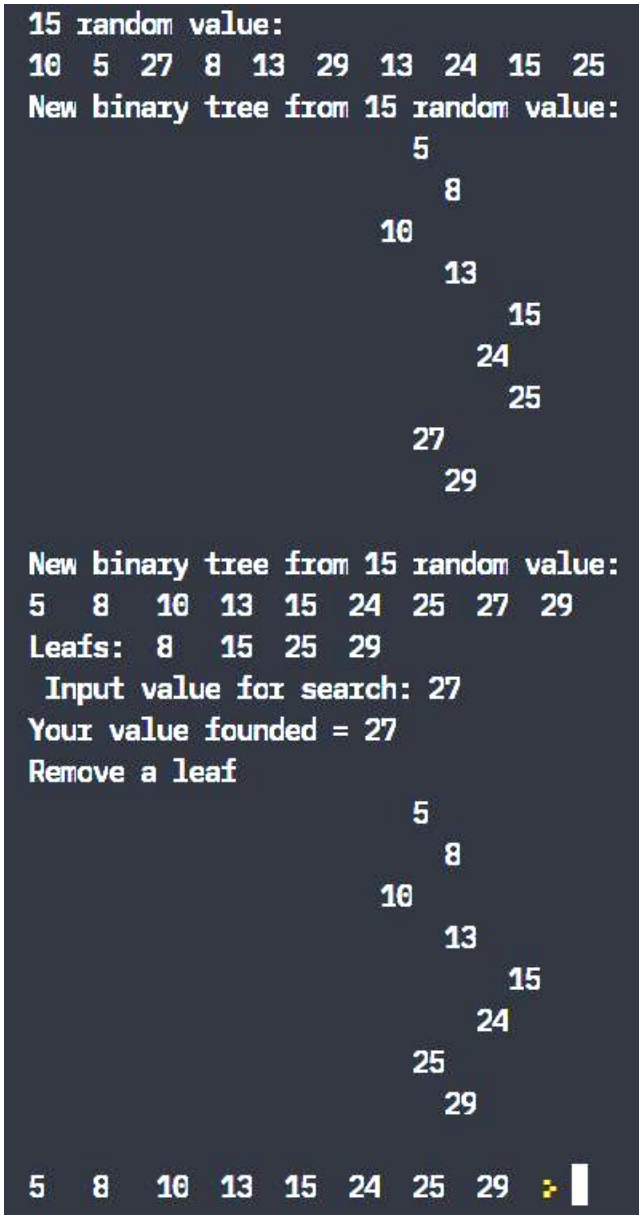
```
  {
   show(Tree->l); // рекурсивний виклик функції для виведення лівого піддерева
   cout << Tree->x << '\t'; // Виведення кореня дерева
   show(Tree->r); // рекурсивний виклик функції для виведення правого піддерева
  }
}
void PrintTree(Node* leaf, int &n) // Функція виведення у формі дерева
{
  if (leaf != NULL)
  {
    n++;
    PrintTree(leaf->l, n);
    for (int i = 1; i <= n; i++) cout << " ";
    cout << leaf->x<<endl;
    PrintTree(leaf->r, n);
    n--;
  }
}
void add_node(int x,Node *&MyTree) //Функція долучення елемента у дерево
{
  if (NULL == MyTree) // Якщо дерева немає, то закладаємо корінь
  {
      MyTree = new Node; // Виділяємо пам'ять під елемент
      MyTree->x = x; // Записуємо дані в елемент
      MyTree->l = MyTree->r = NULL; // гілки порожні
  }
  // Якщо нововведений елемент x менше, ніж елемент дерева, йдемо у ліву гілку
  if (x < MyTree->x)
  {
     // Засобами рекурсії долучаємо елемент на вільне місце
     if (MyTree->l != NULL) add_node(x, MyTree->l);
     else // Якщо елемент отримав своє місце, то
     {
        // виділяємо пам'ять у лівій підгілці. Саме підгілці, а не просто гілці
        MyTree->l = new Node;
        // У лівої підгілки будуть свої ліва і права підгілки, поки вони порожні
        MyTree->l->l = MyTree->l->r = NULL;
        MyTree->l->x = x; // Записуємо елемент у ліву підгілку
     }
   }
   // Якщо нововведений елемент x більше за елемент дерева, йдемо у праву гілку
   if (x > MyTree->x)
   {
      if (MyTree->r != NULL) add_node(x, MyTree->r);
```

```
      else
      {
        MyTree->r = new Node;
        MyTree->r->l = MyTree->r->r = NULL;
        MyTree->r->x = x;
      }
   }
}
// Пошук елемента з певним значенням засобами рекурсії
Node *search(Node *&Tree, int &key)
{
   if (Tree==NULL) return NULL;
   if (key==Tree->x) return Tree;
   else
      if (key<Tree->x) return search(Tree->l,key);
      else return search(Tree->r,key);
}
void Delete(Node*& node, int key); // прототип (реалізація нижче)

//Функція пошуку значення попереднього елемента
void GetPredecessor(Node* node, int& key)
{
  while (node->r != NULL)
  node = node->r;
  key = node->x;
}
void DeleteNode(Node*& node) //Функція видалення елемента
{
  int key;
  Node* ptr = node;
  if (node->l == NULL) { // права гілка
    node = node->r;
    delete ptr;
  }
  else if (node->r == NULL) { // ліва гілка
      node = node->l;
      delete ptr;
    }
    else {
      GetPredecessor(node->l, key);
      node->x = key;
      Delete(node->l, key);
    }
}
```

```
void Delete(Node*& node, int key) // Пошук вузла node, який треба видалити
{
  if (key < node->x)  Delete(node->l, key);
  else
    if (key > node->x)  Delete(node->r, key);
    else   DeleteNode(node);
}
void printLeaf(Node *t)
{
    if ( (t->l == nullptr) && (t->r == nullptr) ) cout << t->x << "\t";
    else
    {
        if (t->l) printLeaf(t->l);
        if (t->r) printLeaf(t->r);
    }
}

void free(Node *&Tree) /* очищення пам'яті*/
{
  if (Tree != NULL)    // Допоки корінь не порожній
  {
    free(Tree->l);      // Рекурсивний виклик для проходу по лівій гілці
    free(Tree->r);      // Рекурсивний виклик для проходу по правій гілці
    delete Tree;        // Видалити корінь дерева
    Tree = NULL;
  }
}
int main()
{
  time_t t;
  srand((unsigned) time(&t));
  Node *Tree = NULL; // Вказівник на вузол (елемент) дерева
  int x,kol=10;
  cout<<"\n15 random value: \n";
  for (int i=0; i<kol; i++)
  {
     x=rand()%31;
     cout<<x<<" ";
     add_node(x,Tree);
  }
  cout<<"\nNew binary tree from 15 random value: \n";
  PrintTree(Tree, kol);
  cout<<"\nNew binary tree from 15 random value: \n";
  show(Tree);
  cout<<"\nLeafs: ";
```

```
  printLeaf(Tree);
  cout << "\n Input value for search: ";cin>>x;
  Node* c=search(Tree, x);
  if(c==NULL) cout<<"Your value not founded! \n";
  else
     cout<<"Your value founded = "<<c->x<<endl;
  cout << "Remove a leaf" << endl;
  Delete(Tree, x);
  PrintTree(Tree, kol);
  cout << endl;
  show(Tree);
  free(Tree);
}
```

Результати:



Контрольні запитання для самоконтролю

1) Що в термінології дерев є висотою дерева?
2) Як в термінології дерев (унікальний) вузол без батька називається?
3) Як в термінології дерев вузол без дітей називається?
4) Чому дорівнює висота порожнього (null)?
5) Які структури даних базуються на бінарному дереві?

Лабораторне завдання

Завдання 1

1. Побудувати бінарне дерево пошуку з вхідної послідовності дійсних чисел. Визначте рівень вузла з мінімальним елементом.

2. Побудувати бінарне дерево пошуку з вхідної послідовності дійсних чисел. Вивести значення вузлів, що лежать на шляху від кореня до вузла з максимальним елементом.

3. Побудувати бінарне дерево пошуку з вхідної послідовності цілих чисел. Визначити значення вузла, дитина якого містить мінімальне значення.

4. Побудувати бінарне дерево пошуку з вхідної послідовності дійсних чисел і визначити його глибину.

5. Побудувати бінарне дерево пошуку з вхідної послідовності дійсних чисел. Вивести значення вузлів, що мають рівень, заданий користувачем.

6. Побудувати бінарне дерево пошуку з вхідної послідовності цілих чисел. Обчислити добуток значень всіх листків.

7. Побудувати бінарне дерево пошуку з вхідної послідовності цілих чисел. Визначити середнє арифметичне значень тих вузлів, які мають рівень, заданий користувачем.

8. Побудувати бінарне дерево пошуку з вхідної послідовності цілих чисел. Визначити суму значень листків з парними значеннями.

9. Побудувати бінарне дерево пошуку з вхідної послідовності цілих чисел. Обчислити середнє арифметичне додатних значень вузлів.

10. Побудувати бінарне дерево пошуку з вхідної послідовності цілих чисел. Визначити кількість листків дерева.

11. Побудувати бінарне дерево пошуку з вхідної послідовності дійсних чисел. Для кожного вузла відображається значення вузла і його рівень зліва направо.

12. Побудувати бінарне дерево пошуку з вхідної послідовності дійсних чисел. Обчислити середнє арифметичне значень всіх вузлів. Вивести значення вузлів дерева, які більше середнього арифметичного.

13. Побудувати бінарне дерево пошуку з вхідної послідовності цілих чисел. Обчислити суму двоцифрових вузлів. Вивести значення вузлів дерева, які більше середнього арифметичного.

14. Побудувати бінарне дерево пошуку з вхідної послідовності цілих чисел. Визначити суму значень листків з непарними значеннями.

15. Побудувати бінарне дерево пошуку з вхідної послідовності цілих чисел. Вивести значення вузлів дерева, кратних 3, а також номери рівня для кожного такого вузла.

16. Побудувати бінарне дерево пошуку з вхідної послідовності символів. Вивести на екран вузли, що містять голосні літери, та вказати їхні рівні.

17. Побудувати бінарне дерево пошуку з вхідної послідовності цілих чисел. Вивести всіх дітей трицифрових вузлів.

18. Побудувати бінарне дерево пошуку з вхідної послідовності цілих чисел. Визначити суму значень тих вузлів, які лежать на шляху від кореня до мінімального елементу.

19. Побудувати бінарне дерево пошуку з вхідної послідовності дійсних чисел. Обчислити добуток одноцифрових вузлів.

20. Побудувати бінарне дерево пошуку з вхідної послідовності дійсних чисел. Визначити рівень вузла з максимальним елементом.

21. Побудувати бінарне дерево пошуку з вхідної послідовності цілих чисел. Знайти лист з максимальним значенням.

22. Побудувати бінарне дерево пошуку з вхідної послідовності символів. Вивести на екран всі символи арифметичних дій та їхні рівні.

23. Побудувати бінарне дерево пошуку з вхідної послідовності цілих чисел. Вивести значення всіх листів дерева.

24. Побудувати бінарне дерево пошуку з вхідної послідовності дійсних чисел. Визначити рівень вузла, дитина якого є максимальним значенням.

25. Побудувати бінарне дерево пошуку з вхідної послідовності дійсних чисел. Обчислити середньоарифметичне максимального і мінімального вузлів.

Завдання 2*

Ввести число. Визначити, чи присутнє це число в дереві, створеному у Завданні 1. Видалити знайдений елемент. Продемонструвати для листка, вузла з одним піддеревом, вузла з двома піддеревами.

Самостійна робота 2
Бінарні дерева

Індивідуальні завдання для самостійної роботи

1. Створити шаблон класу бінарного дерева пошуку з такими операціями.
   1.1. долучення (вставлення) елемента в дерево;
   1.2. видалення елемента з дерева;
   1.3. пошук елемента в дереві. Створити два варіанти функції: перша функція повертає значення true, якщо елемент є в дереві, false – якщо його немає; друга функція повертає вказівник на елемент, якщо він є в дереві, і nullptr якщо його там немає.
   1.4. пошук мінімального і максимального елементів дерева;
   1.5. визначення кількості елементів у дереві;
   1.6. визначення висоти дерева;
   1.7. пошук батька елемента;
   1.8. прямий, зворотний, центрований обхід дерева;
   1.9. обхід дерева в ширину;
   1.10. формування лінійного дужкового запису дерева;
   1.11. видалення дерева.

2*. На основі класу бінарного дерева реалізовувати клас «множина», аналогічний класу set стандартної бібліотеки шаблонів C++. Операції класу «множина»:
   2.1. конструктор, що створює пусту множину;
   2.2. конструктор, що створює множину з заданим критерієм сортування;
   2.3. конструктор копіювання;
   2.4. конструктор, що створює множину, ініціалізовану елементами інтервалу [begin, end);
   2.5. конструктор, що створює множину, ініціалізовану елементами інтервалу [begin, end) з заданим критерієм сортування;
   2.6. визначення кількості елементів множини;
   2.7. перевірка, чи є множина порожньою;
   2.8. перевірка, чи входить елемент у множину.

3. На основі класу бінарного дерева реалізувати клас збалансованого дерева. Для балансування використовувати додатковий відсортований масив і бінарний пошук.

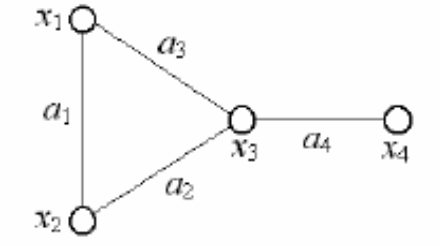
4. На основі класу бінарного дерева реалізувати клас збалансованого дерева. Для балансування використовувати алгоритм DSW.

Лабораторна робота 7
Графи

Теоретичні відомості

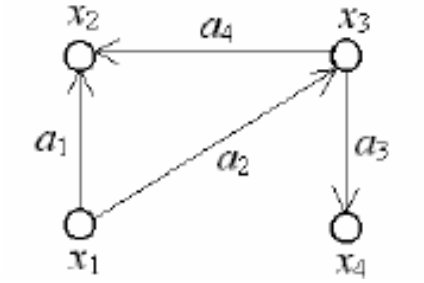
Існує два основних типи графів: орієнтований і неорієнтований. Якщо ребра графа задані напрямками від однієї вершини до іншої, то такий граф називається орієнтованим (орграф).

Приклад неорієнтованого графа *G = (X, E)*:



X = {*x1*, *x2*, *x3*, *x4*},
E = {*a1*(*x1*, *x2*), *a2*(*x2*, *x3*), *a3*(*x1*, *x3*), *a4*(*x3*, *x4*)}.

Приклад орієнтованого графа *G = (X, A)*:



X = {*x1*, *x2*, *x3*, *x4*},
A = {*a1*(*x1*, *x2*), *a2*(*x1*, *x3*), *a3*(*x3*, *x4*), *a4*(*x3*, *x2*)}.

Ребра орграфа ще називають дугами. Відповідні вершини орієнтованого графа називаються початком і кінцем.

Якщо напрямки ребер не вказані, то граф називається **неорієнтований** або просто графом (н-граф, мережа).

Інцидентними ребру вершинами, називаються вершини, з'єднані ребром. Дві вершини називаються **суміжними**, якщо вони інцидентні одному і тому самому ребру.

Два ребра називаються **суміжними**, якщо вони мають загальну вершину.

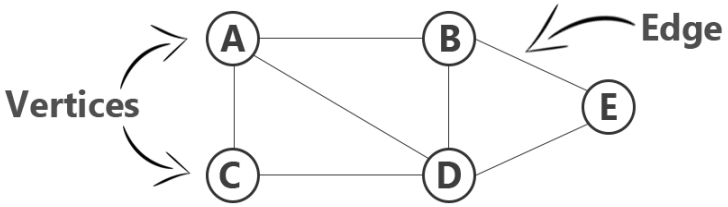
Граф називається **простим**, якщо кожну пару вершин з'єднує не більше, ніж одне ребро.

Граф називається **мультиграфом**, якщо він має кратні ребра.

Граф називається **псевдографом**, якщо він має петлі та кратні ребра.

Будь-якому ребру відповідає рівно дві вершини, а вершині може відповідати довільне число ребер, це число визначає **степінь вершини**. Степеню вершини н-графа називається число ребер, інцидентних цій вершині.

Приклад графа з 5 вершинами і 6 ребрами:



Цей граф G можна визначити як G = (V, E), де V = {A,B,C,D,E} і E = {(A,B),(A,C)(A,D),(B,D),(C,D),(B,E),(E,D)}.

Шлях - це послідовність вершин (без повторів), в якій будь-які дві сусідні вершини суміжні. Наприклад, у графі є різні *шляхи* з *вершини* A до *вершини* E: ABE, ADE, ADBE, ACDE тощо.

Довжина шляху - кількість ребер, з яких складається цей шлях. Наприклад, довжина вже згаданих шляхів ABE і ACDE – 2 і 3 відповідно. Відстань між вершинами – це довжина найкоротшого шляху між ними. З цього визначення видно, що відстань між вершинами A і E на графі дорівнює 2.

Приклади програм

Приклад 1. Обхід вшир. Для неорієнтованого графа найти відстань від одної заданої вершини до іншої.

Для розв'язання цієї задачі варто використовувати алгоритм «Пошук вшир». Суть даного алгоритму полягає в тому, що всі вершини, починаючи з початкової, поміщаються в структуру черга (queue) в порядку віддалення від початкової вершини. По мері заповнення черги, кожній вершині приписується значення відстані (dist) от початкової вершини, після чого відповідна вершина позначається як пройдена (used) і її відстань від початкової вершини більше не переписується навіть при повторному перегляді. Отже, кожній вершині, для якої є шлях, що з'єднує її з початковою вершиною, співставляється мінімальна відстань від неї до початкової вершини. Якщо такого шляху не існує, відстань залишається нульовою.

```
#include <iostream>
using namespace std;
 struct node
{ int value;
  node * next;
  node(int v, node * n = NULL)
  {   value = v;
      next = n;
} };
```

```
struct queue // Структура черга
{
  node * start;
  node * finish;
  int len = 0;
  queue(node * s = NULL, node * f = NULL)
  {
     start = s;
     finish = f;
  }
  string push(int n)
  {
     if(len == 0)
     {
        node * new_node = new node(n,finish);
        start = new_node;
        finish = new_node;
     }
     else
     {
        node * new_node = new node(n,NULL);
        start -> next = new_node;
        start = new_node;
     }
     len++;
     return "ok";
  }
  int pop()
  {
     int v = finish -> value;
     finish = finish -> next;
     len--;
     return v;
    }
  int front()
  {
     return finish -> value;
  }
  int size()
  {
     return len;
  }
};
```

```
int main()
{
  int n, s, f;
  cout<<"Введіть кількість вершин у графі n: ";cin >> n;
  cout<<"Введіть номер першої вершини у графі s (s<n): ";cin >> s;
  cout<<"Введіть номер останньої вершини у графі f (s<f<n): ";cin >> f;
  int graph[n][n];
  int dist[n]; // Масив відстаней від початкової до i-ої вершини
  bool used[n];
  /* Далі в n рядках задають матрицю суміжності. Якщо значення в j-м елементі i-й рядка
  дорівнює 1, то в графі є напрямлене ребро з вершини i у вершину j. */
  cout<<"Введіть матрицю суміжності графа (матриця з 1 і 1 розміром n x n):\n ";
  for(int i = 0; i < n; i++)
  for(int j = 0; j < n; j++)
      cin >> graph[i][j];
  for(int i = 0; i < n; i++)
  {
      dist[i] = 0;
      used[i] = false;
  }
  queue vertex; // Черга вершин
  vertex.push(s-1);
  used[s-1] = true;
  while(vertex.size() != 0)
  {
    int v = vertex.front();
    for(int i = 0; i < n; i++)
    {
      if(graph[v][i] == 1 && used[i] == false)
      {  vertex.push(i);
        dist[i] = dist[v] + 1;
        used[i] = true; // i-а вершина пройдена і не буде коригуватися пізніше
       }
     }
    used[v] = true;
    vertex.pop();
  }
  if(dist[f-1])
       cout<<"Мінімальна відстань від початкової вершини до кінцевої = "<< dist[f-1];
  else cout <<"Шляху не існує, відстань = "<<dist[f-1];
  return 0;
}
```

Результати:

```
Введіть кількість вершин у графі n: 5
Введіть номер першої вершини у графі s (s<n): 1
Введіть номер останньої вершини у графі f (s<f<n): 5
Введіть матрицю суміжності графа (матриця з 0 і 1 розміром n x n):
 0  1  0  0  0
1  0  1  0  0
0  1  0  1  1
0  0  1  0  0
0  0  1  0  0
Мінімальна відстань від початкової вершини до кінцевої = 3
```

```
Введіть кількість вершин у графі n: 6
Введіть номер першої вершини у графі s (s<n): 2
Введіть номер останньої вершини у графі f (s<f<n): 5
Введіть матрицю суміжності графа (матриця з 1 і 1 розміром n x n):
 0  1  1  1  1  1
1  0  0  1  1  1
1  1  0  0  1  0
1  1  0  0  0  0
1  1  1  0  0  1
1  1  0  0  1  0
Мінімальна відстань від початкової вершини до кінцевої = 1
```

```
Введіть кількість вершин у графі n: 4
Введіть номер першої вершини у графі s (s<n): 1
Введіть номер останньої вершини у графі f (s<f<n): 4
Введіть матрицю суміжності графа (матриця з 0 і 1 розміром n x n):
 0  1  0  0
1  0  1  0
0  1  0  1
0  0  1  0
Шляху не існує, відстань = 0
```

Приклад 2. Обхід вглиб. Ввести кількість вершин графа і сам неорієнтований ненавантажений граф, в якому вибрати вершину. Визначити шляхи до кожної з решти вершин.

```
#include <iostream>
#include <queue>
using namespace std;
int main()
{
  int n, v;
  cout<<"Введіть кількість вершин у графі n: ";  cin >> n;
```

```
 cout<<"Введіть номер вершини у графі v (v<n): "; cin >> v;
 /*Далі в n рядках задають матрицю суміжності. Якщо значення в j-м елементі i-й рядка
 дорівнює 1, то в графі є ребро з вершини i у вершину j. Гарантується, що на головній
 діагоналі матриці завжди стоять нулі.*/
 cout<<"Введіть матрицю суміжності графа (матриця з 0 і 1 розміром n x n):\n ";
 int M[n][n];
 for (int i = 0; i < n; i++)
 {
   for (int j = 0; j < n; j++) cin >> M[i][j];
   M[i][i] = -1;
 }
 queue <int> plan; // план перебору у вигляді черги
 plan.push(--v); // ми нумеруємо з 0, а не з 1
 M[v][v] = 0; // позначаємо, що ця вершина вже заносилась у план
 while (!plan.empty())
 {
   v = plan.front(); // беремо наступну по плану вершину
   plan.pop(); // видаляємо її з плану перебору
   for (int u = 0; u < n; u++) // перебираємо сусідні з нею
     if (M[v][u] == 1 and M[u][u] == -1)  // якщо нова, то
     {
        plan.push(u); // додаємо її в план
        M[u][u] = M[v][v] + 1; // шлях на 1 крок довше
     }
 }
 cout<<"Відстані до i-х елементів - це відстані від заданої вершини до i-ої"<<endl;
 for (int u = 0; u < n; u++) cout << M[u][u] << " ";
}
```

Результати:

```
Введіть кількість вершин у графі n: 5
Введіть номер вершини у графі v (v<n): 1
Введіть матрицю суміжності графа (матриця з 0 і 1 розміром n x n):
 0   1   1   0   1
1   0   1   1   0
1   1   0   0   0
0   1   0   0   0
1   0   1   0   0
Масив відстаней, де i-й елемент - це відстань від початкової вершини до  i-ої
0 1 1 2 1
```

Приклад 3. Обхід вглиб. Дано неорієнтований ненавантажений граф, в якому вибрана вершина. Найти кількість вершин, які лежать з нею в одній компоненті зв'язності (включаючи саму вершину).

```
#include <iostream>
#include <stack>
using namespace std;
int main()
{
  int n, s, counter=1;
  cout<<"Введіть кількість вершин у графі n: ";   cin >> n;
  cout<<"Введіть номер виділеної вершини у графі s (s<n): ";  cin >> s;
  s--;
  /* Далі в n рядках задають матрицю суміжності. Якщо значення в j-м елементі i-й рядка дорівнює 1, то в графі є ребро з вершини i у вершину j.Гарантується, що на головній діагоналі матриці завжди стоять нулі.*/
  cout<<"Введіть матрицю суміжності графа (матриця з 0 і 1 розміром n x n):\n ";
  int matrix[n][n];
  for(int i=0;i<n;i++)
  for(int j=0;j<n;j++)  cin>>matrix[i][j];
  // Перевіряємо рядок s і записуємо в стек всі вершини інцидентні даній.
  stack<int> st;
  for(int j=0; j<n; j++)
     if(matrix[s][j] == 1) st.push(j);
  /* Оскільки в умові гарантується наявність на головній діагоналі нулів, то будемо позначати перевірені вершини за допомогою елемента, розташованого на головній діагоналі (тобто присвоїмо йому значення, відмінне від 0, наприклад, 1).*/
 matrix[s][s]=1;
 /* Перевірятимемо усі рядки матриці в стеку до його опорожнення, і збільшуватимемо лічильник на одиницю після видалення зі стеку незначеної вершини.*/
 while(!st.empty())
 {
      int a=st.top();
      st.pop();
      if(matrix[a][a]!=1)
      {
         for(int j=0;j<n;j++)
            if(matrix[a][j] == 1) st.push(j);
         counter++;
         matrix[a][a]=1;
  }  }
 cout<<counter<<endl;
}
```

Результати:

```
Введіть кількість вершин у графі n: 4
Введіть номер виділеної вершини у графі s (s<n): 1
Введіть матрицю суміжності графа (матриця з 0 і 1 розміром n x n):
 0  1   0   0
1   0   1   0
0   1   0   1
0   0   1   0
Кількість вершин, які лежать з нею в одній компоненті зв'язності: 4
```

Контрольні запитання для самоконтролю

1) Що таке граф?
2) Чим орієнтовані графи відрізняються від неорієнтованих?
3) Що таке мультиграф?
4) Що таке псевдограф?
5) Який граф називається простим?
6) Що називають інцидентністю ребра графа?
7) Що таке степінь вершини графа?
8) Що таке шлях графа?
9) Як визначається довжина шляху?
10) Навести приклади застосування графів.

Лабораторне завдання

1. Записати матрицю суміжності для заданого графа.

2. Написати програму, яка для заданого графа створює список суміжності по матриці суміжності з п. 1.

3. Зобразити граф за заданою матрицею суміжності.

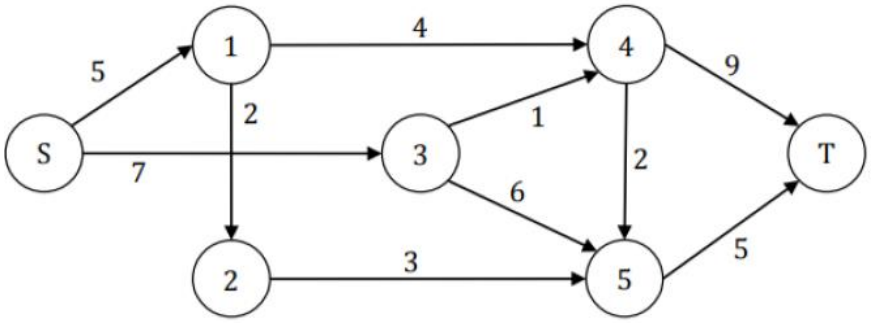
4. Виконати ручний розрахунок алгоритму пошуку найкоротшого шляху в заданому графі. Написати програму пошуку найкоротшого шляху і протестувати її для графа, заданого в індивідуальному варіанті.

5. Виконати ручний розрахунок алгоритмів обходу заданого графа в ширину та глибину. Написати програму обходу графа в ширину і глибину і протестувати її для графа, заданого в індивідуальному варіанті.

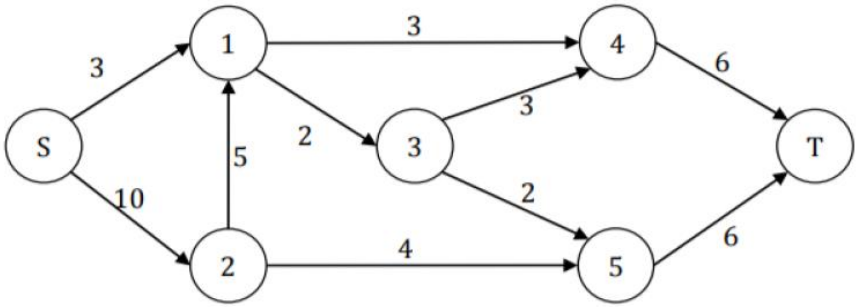
Індивідуальні варіанти

Варіанти для завдань 1-2, 4, 5

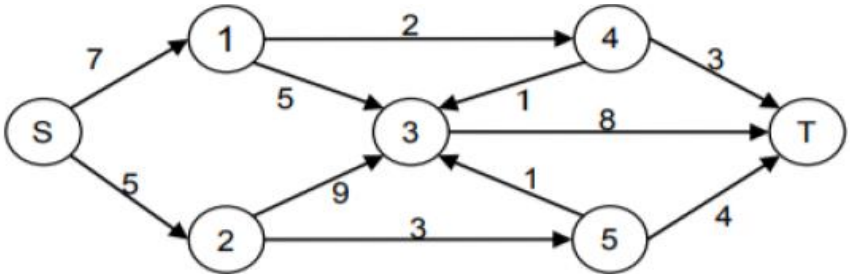
Варіант 1



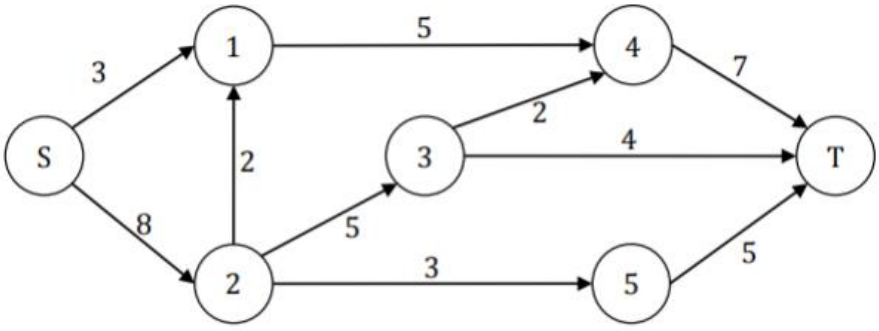
Варіант 2



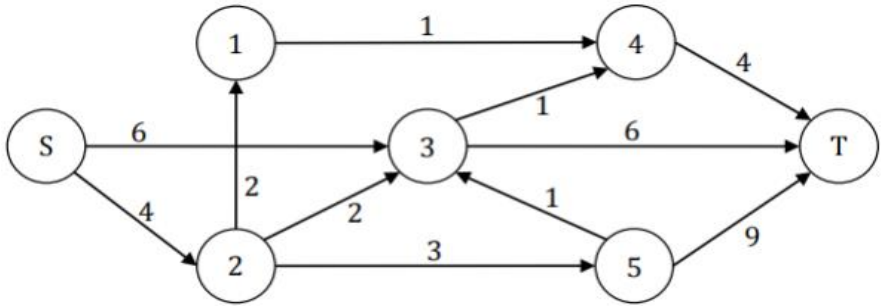
Варіант 3



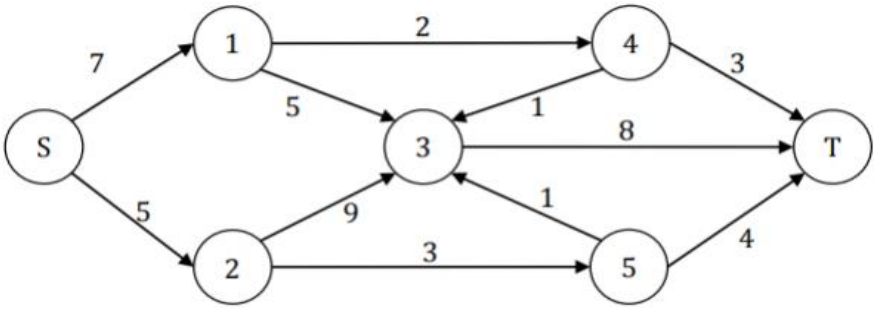
Варіант 4



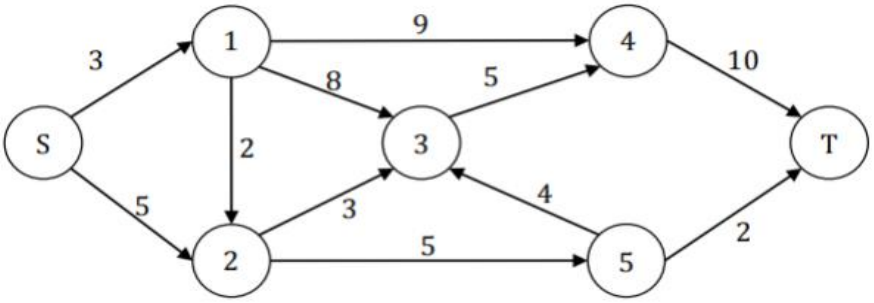
Варіант 5



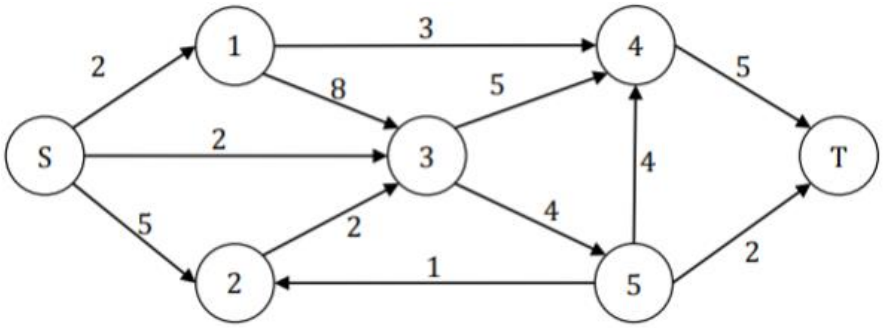
Варіант 6



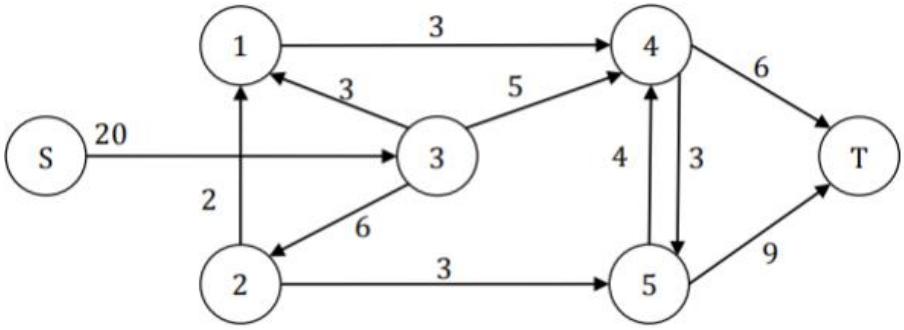
Варіант 7



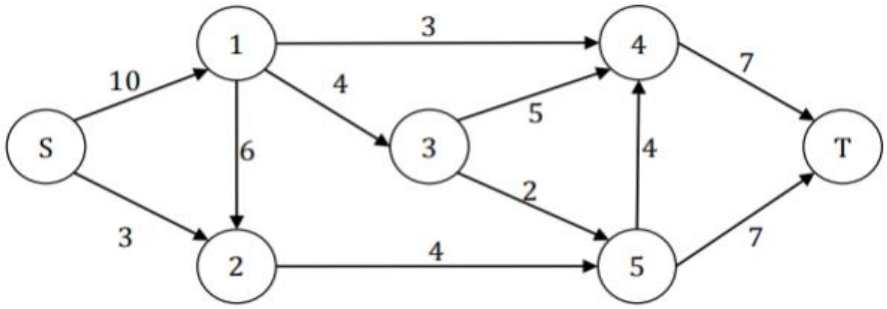
Варіант 8



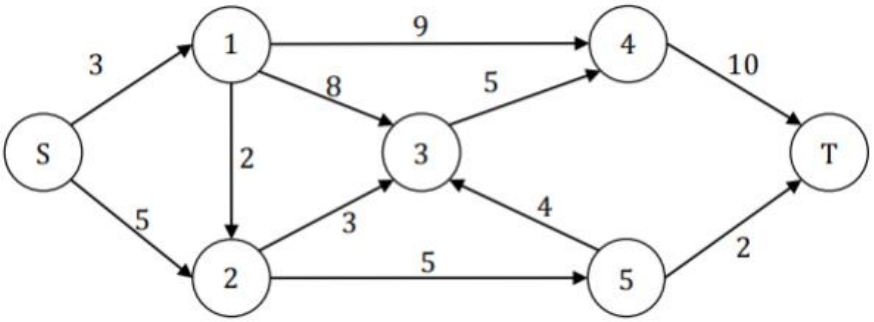
Варіант 9



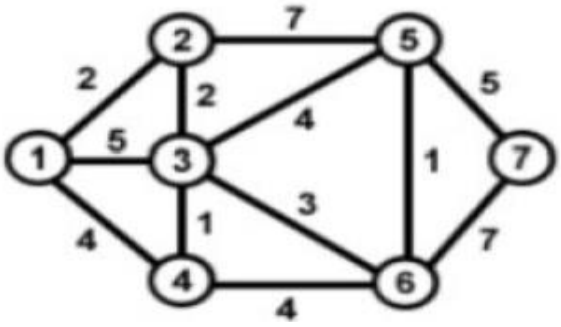
Варіант 10



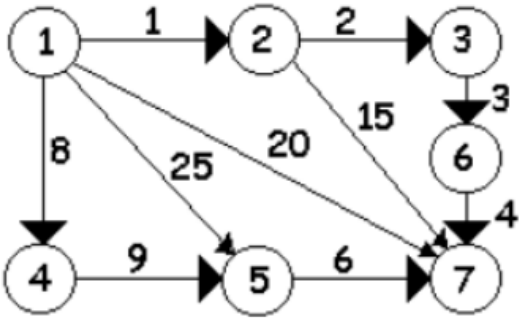
Варіант 11



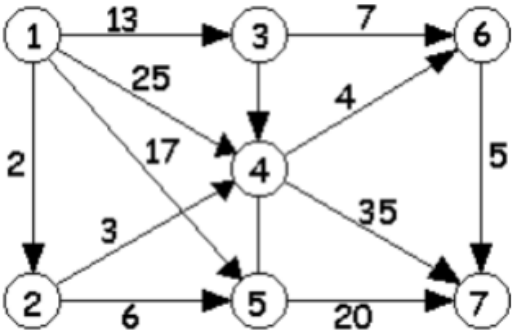
Варіант 12



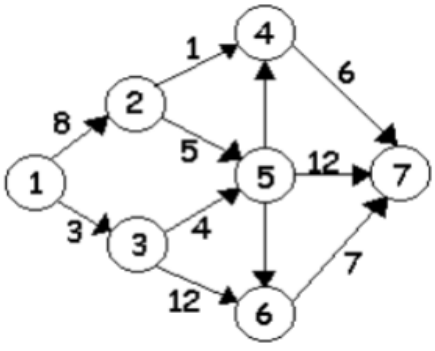
Варіант 13



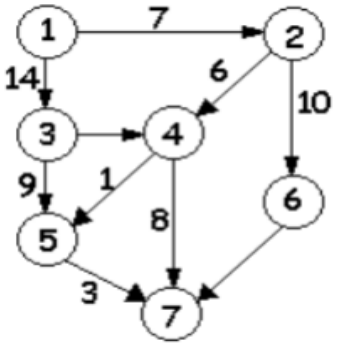
Варіант 14



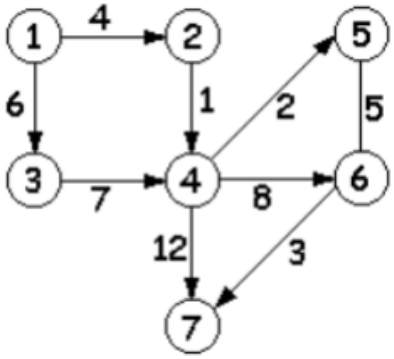
Варіант 15



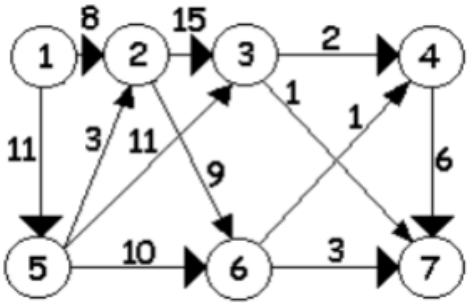
Варіант 16



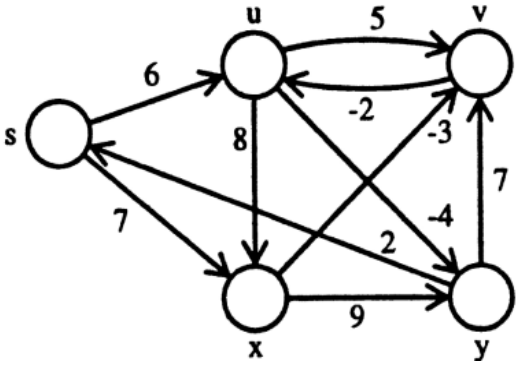
Варіант 17



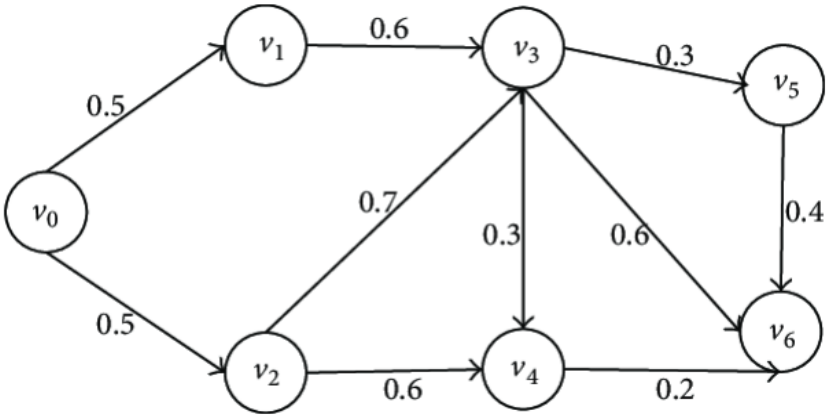
Варіант 18



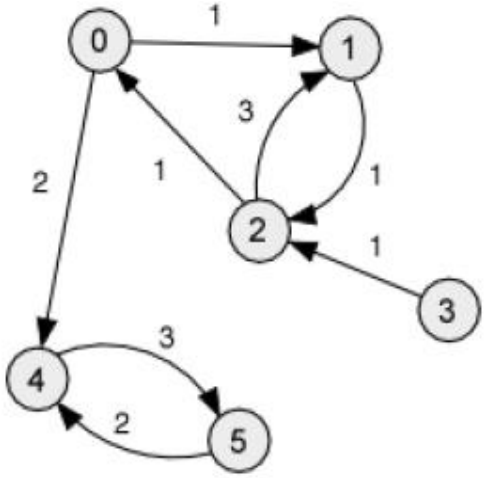
Варіант 19



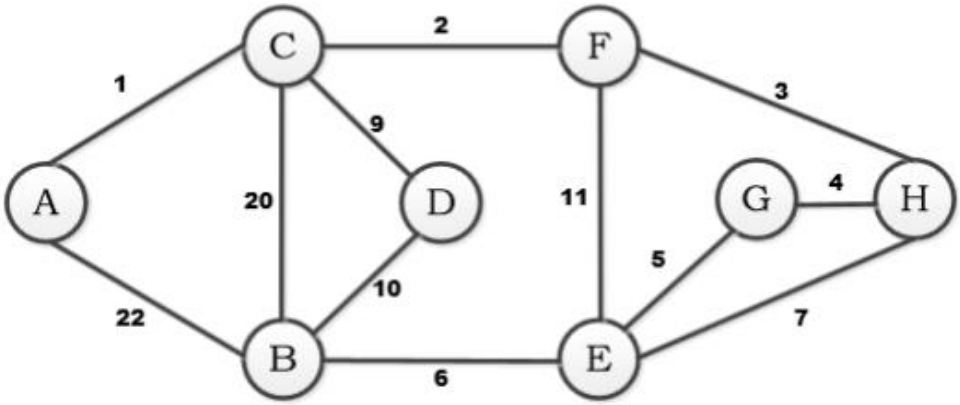
Варіант 20



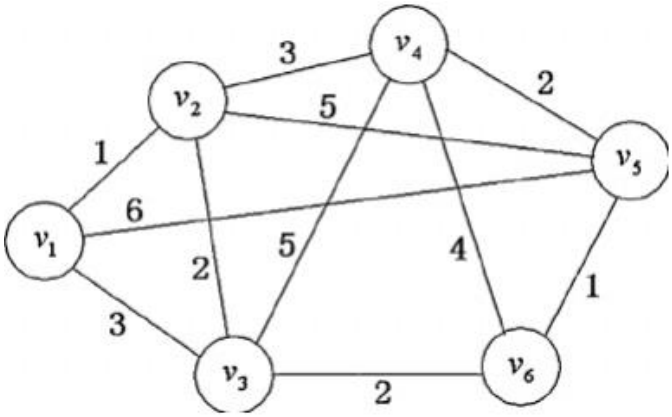
Варіант 21



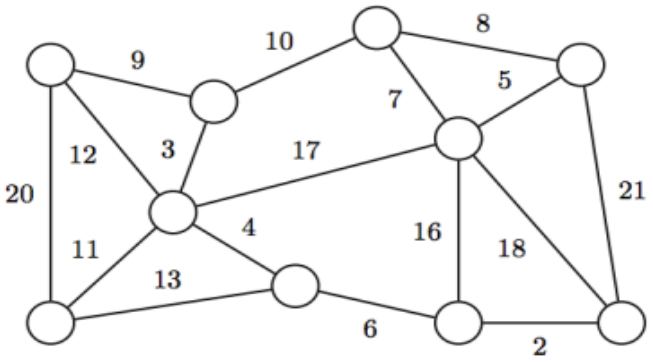
Варіант 22



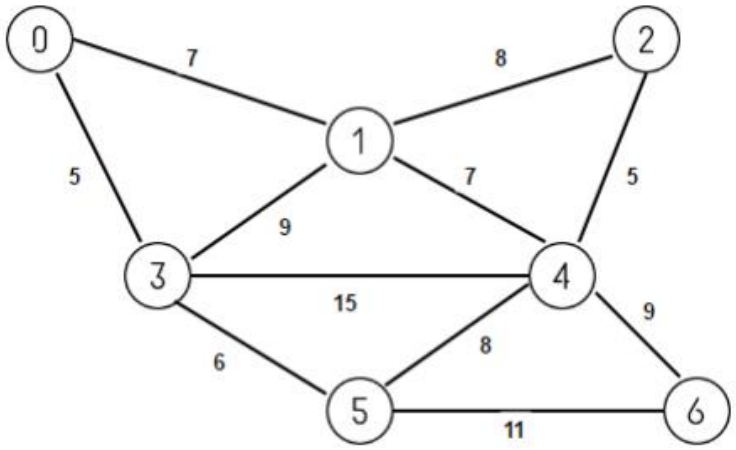
Варіант 23



Варіант 24



Варіант 25



Варіанти для завдання 3

Варіант 1

	1	2	3	4	5	6	7	8
1		5	6		3	5		2
2	5			9		5	5	5
3	6			5	6	1	4	8
4		9	5		2	1	3	
5	3		6	2			9	7
6	5	5	1	1			3	8
7		5	4	3	9	3		
8	2	5	8		7	8		

Варіант2

	1	2	3	4	5	6	7	8
1		1		9		1		3
2	1		4		6	9		5
3		4		6	5		9	
4	9		6		9	4		4
5		6	5	9		1	9	6
6	1	9		4	1			
7			9		9			2
8	3	5		4	6		2	

Варіант 3

	1	2	3	4	5	6	7	8
1		2		8	6			3
2	2		2	6		7		3
3		2		9	6	4	1	
4	8	6	9				5	3
5	6		6			7	1	4
6		7	4		7		3	
7			1	5	1	3		6
8	3	3		3	4		6	

Варіант 4

	1	2	3	4	5	6	7	8
1		7	9			1		
2	7		2	8		2	1	7
3		2		2	8		4	5
4		8	2		3	1		
5	9		8	3		8	4	1
6	1	2		1	8		7	9
7		1	4		4	7		1
8		7	5		1	9	1	

Варіант 5

	1	2	3	4	5	6	7	8
1			6	3			7	1
2			9	2		1	7	9
3	6	9		6	1			9
4	3	2	6		8	9		
5			1	8		6	2	
6		1		9	6		6	7
7	7	7			2	6		9
8	1	9	9			7	9	

Варіант 6

	1	2	3	4	5	6	7	8
1		2	9		6		2	8
2	2			2		4	8	2
3	9			7	4	4	3	
4		2	7		6		2	9
5	6	5		6		3	9	2
6		4	4		3		4	
7	2	8	3	2	9	4		3
8	8	2		9	2		3	

Варіант 7

	1	2	3	4	5	6	7	8
1		3	8		6		7	9
2	3		1		6			3
3	8	1		4		8		
4			4		9	1	9	4
5	6	6		9			5	8
6			8	1			8	
7	7			9	5	8		9
8	9	3		4	8		9	

Варіант 8

	1	2	3	4	5	6	7	8
1		5		6		6		
2	5		7		8		1	5
3		7		7	5	5	7	
4	6		7			9		7
5		8	5				8	6
6	6		5	9				
7		1	7		8			2
8		5		7	6		2	

Варіант 9

	1	2	3	4	5	6	7	8
1		6	8		4		3	
2	6		7	1		8	6	4
3	8	7		9	5	2		
4		1	9		7		1	
5	4		5	7		7		4
6		8	2		7		5	3
7	3	6		1		5		
8		4			4	3		

Варіант 10

	1	2	3	4	5	6
1	5				2	
2			3	1		4
3		1		2		
4		5				
5	2			6		2
6			6		3	

Варіант 11

	1	2	3	4	5	6
1				4		
2			3			
3	4					4
4		5		1		1
5			1	2		
6	4		6		5	

Варіант 12

	1	2	3	4	5	6
1	4		4			2
2			1			
3	4				3	
4		6				
5	6			3		3
6		2		6		

Варіант 13

	1	2	3	4	5	6
1	2				2	
2		3		1		
3				5		2
4	6	4				6
5		5			3	
6	1			2		5

Варіант 14

	1	2	3	4	5	6
1		4	3			
2				1		
3	6					5
4	5		2			
5			1		1	
6		6		1		3

Варіант 15

	1	2	3	4	5	6
1		4	5			4
2	4					
3		6			3	
4		5		1		3
5			1			
6	5				6	

Варіант 16

	1	2	3	4	5	6
1	2				2	
2		5		3		2
3	2			2		
4			2			
5						6
6	2			6		

Варіант 17

	1	2	3	4	5	6
1			3			
2	2				8	
3				5		1
4		6			4	
5	4					3
6			4			

Варіант 18

	1	2	3	4	5	6	7	8
1		5	6		3	5		2
2	5			3		5	5	
3	6				6		0	8
4		3			2	1	3	
5	3		6	2			9	7
6	5	5		1			3	8
7		5	0	3	9	3		
8	2		8		7	8		

Варіант 19

	1	2	3	4	5	6	7	8
1		2		9		1		3
2	2		2			7		5
3		2		6	5		9	
4	9		6		9			4
5			5	9		1	9	6
6	1	7			1			
7			9		9			2
8	3	5		4	6		2	

Варіант 20

	1	2	3	4	5	6	7	8
1		2		3	6			3
2	2		2			7		3
3		2		9	6	4	1	
4	3		9					3
5	6		6			7	1	
6		7	4		7		3	
7			1		1	3		6
8	3	3		3			6	

Варіант 21

	1	2	3	4	5	6	7	8
1		7	9			1		
2	7		2			2		7
3		2		2	8		4	
4			2		4	1		
5	9		8	4		8	4	1
6	1	2		1	8		7	10
7			4		4	7		1
8		7			1	10	1	

Варіант 22

	1	2	3	4	5	6	7	8
1		1		3			7	1
2	1		9	2		1	3	
3		9		6	1			9
4	3	2	6		8	9		
5			1	8		6	2	
6		1		9	6		6	7
7	7	3			2	6		9
8	1		9			7	9	

Варіант 23

	1	2	3	4	5	6	7	8
1			9		6		2	8
2			3	2		3		2
3	9	3		7	4	4	3	
4		2	7		6			9
5	6			6		3	7	2
6		3	4		3		4	
7	2		3		7	4		3
8	8	2		9	2		3	

Варіант 24

	1	2	3	4	5	6	7	8
1		2	8		4		7	1
2	2		1		6			3
3	8	1		4		8		
4			4		9	3		4
5	4	6		9			5	8
6			8	3			8	
7	7				5	8		9
8	1	3		4	8		9	

Варіант 25

	1	2	3	4	5	6	7	8
1		7		6		6		
2	7		9		8		1	5
3		9		7	2	4	7	
4	6		7			9		7
5		8	2				8	
6	6		4	9				
7		1	7		8			2
8		5		7			2	

Самостійна робота 3
Автомати

Теоретичні відомості

1 Поняття автомата

Автомат – це абстрактний об'єкт, призначений для виконання певної роботи. Саме теорія автоматів застосовується для шифрування текстів і в теорії алгоритмів. Також автомати використовуються для описування поведінки різноманітних систем керування, таких як цифрові схеми, протоколи обчислювальних систем тощо. Фактично за допомогою автомата можна реалізувати перекладання текстів з однієї мови на іншу. Певним продовженням ідеї автоматів є поняття найважливішої частини комп'ютера – процесора (це автомат з пам'яттю), який перетворює одну двійкову послідовність на іншу.

Кожен автомат отримує для опрацювання вхідну послідовність даних (сигналів), а як результат видає вихідну послідовність даних (сигналів).

Кожен автомат має скінчену множину станів. Автомат переходить з одного стану до іншого під впливом послідовності літер вхідного алфавіту.

Алфавіт автомата – це скінчена множина певних символів. Наприклад, множина `{1, 2}` – це алфавіт, який складається з одиниці та двійки, множина `{A, B, …, Z}` – це алфавіт великих латинських літер.

Вхідним алфавітом є скінчена множина допустимих вхідних символів, з якої формуються зчитуванні автоматом рядки. *Вихідний алфавіт* – це скінчена множина допустимих вихідних символів, з якої формуються рядки-результати. Найпоширенішим є різновид автоматів, коли набори елементів вхідного й вихідного алфавітів автомата збігаються.

За приклад доволі простого автомата розглянемо ліфт у двоповерховому будинку. Припустімо, що цей ліфт вміє реагувати на такі *команди*:

- `O` – відчинити двері (від англ. open – відчинити);
- `C` – зачинити двері (від англ. close – зачинити);
- `D` – спуститися на перший поверх (від англ. down – донизу);
- `U` – піднятися до другого поверху (від англ. up – догори).

Окрім того, ліфт може виконувати комбінацію вище наведених команд, наприклад: "`CUO`", що означає: "зачинити двері", "піднятися до другого поверху", "відчинити двері". Але існують команди, які для такого ліфта будуть некоректними. Приміром, не можна рухатися з відчиненими дверима чи підніматися на другий поверх, коли ліфт уже перебуває на другому поверсі.

Тепер наведемо чотири можливі *стани* ліфта:

- `CD` – перший поверх, двері зачинено;

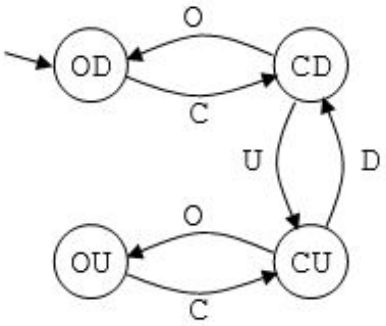
– OD – перший поверх, двері відчинено;
– CU – другий поверх, двері зачинено;
– OU – другий поверх, двері відчинено.

Вважатимемо, що спочатку ліфт (автомат) перебуває на першому поверсі, його двері є зачинено, тобто початковий стан ліфту є "CD".

Завдання полягає у тому, щоб змоделювати такий ліфт на комп'ютері, тобто написати програму, спроможну виконувати команди ліфта (чи то повідомляти, що команда є некоректна).

Кожна з команд переводитиме ліфт із одного стану до іншого, приміром, якщо ліфт перебував у стані "CU" і надійшла команда "D", то ліфт перейде до стану "CD".

Переходи з одного стану до іншого можна описати графом переходів автомата, вершинами якого будуть стани автомата, а дугами – можливі переходи кожного такту роботи. Усі можливі команди і стани ліфта можна графічно зобразити так:



На цій схемі чотирма кружечками позначено стани автомата (початковим станом є стан "OD" – перший поверх, двері відчинено), дугами зі стрілками – переходи з одного стану до іншого, текстовим підписом біля дуг – команди, виконувані ліфтом. Наприклад, зі стану "OU" за команди "C" автомат перейде до стану "CU".

Ідентичним до графічного поданням автомата є так звана "таблиця правил" (чи "таблиця переходів"):

Стан	**Команди**			
	O	**C**	**U**	**D**
CD	OD	–	CU	–
OD	–	CD	–	–
CU	OU	–	–	CD
OU	–	CU	–	–

Якщо ліфт перебуває у стані "CD" і надійшла команда "O", то ліфт відчинить двері і, згідно до таблиці, перейде до стану "OD" (перший поверх, двері відчинено).

Отже розглянутий автомат здатен опрацьовувати команди: "O", "C", "U", "D". Ця множина формує алфавіт наведеного автомата, тобто набір можливих опрацьовуваних автоматом даних: {O, C, D, U}.

З алфавітних символів можна формувати рядки, приміром, “CUO” означає три послідовні команди: “зачинити двері”, “піднятися до другого поверху” і “відчинити двері”. Для виконання цих трьох команд слід застосувати автомат тричі. За першого застосування автомат стартує з початкового стану “OD”. Після першого перетворення автомат змінить стан: зі стану “OD” дугою “C” здійсниться перехід до стану “CD”. Після другого переходу зі стану “CD” за команди “U” автомат змінить свій стан на “CU”. Після третього переходу за команди “O” автомат перейде до стану “OU”.

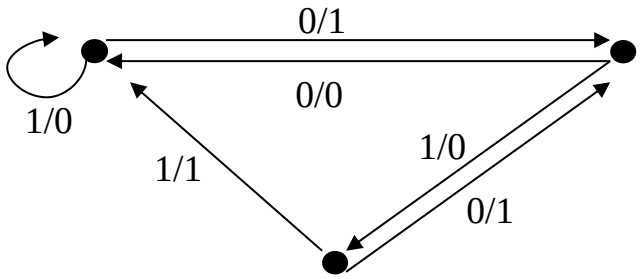
2 Синхронні автомати

Автомат називають *синхронним*, якщо він перетворює один елемент вхідного на один елемент вихідного алфавіту.

Програми, що реалізують синхронні автомати, можуть використовувати оператори циклу `for` для опрацювання послідовності елементів та вкладені в нього оператори `switch` та `if` для аналізу й перетворювання елементів.

Приклади програми

Приклад 1. Автомат задано алфавітом X = Y ={0, 1} і трьома станами – A, B, C:



Ввести вхідну послідовність символів S з алфавітом X і здобути вихідну послідовність після дії заданого автомата (перетворений рядок S).

Розв’язок. По дугах автомата можна визначити, наприклад, що в стані A елемент 0 буде перетворено на 1 і при цьому автомат перейде до стану B, а елемент 1 перетвориться на 0 і автомат залишиться у тому ж самому стані A. Аналогічно інтерпретуються інші дуги. Цей автомат, заданий графічно, можна подати також у вигляді таблиці переходів:

Стан	Вхідний елемент	
	0	**1**
A	B,1	A,0
B	A,0	C,0
C	B,1	A,1

У наведеній програмі опрацювання вхідної послідовності реалізовано у функції. Зауважимо, що цю функцію можна реалізувати багатьма іншими способами. Стан автомата (`A`, `B` чи `C`) обирається послідовно для кожного з опрацьовуваних символів рядка за допомогою оператора `switch`.

На вхід автомата подається рядок символів з вхідного алфавіту. Кожен з цих символів буде перетворено згідно до команд переходу, при цьому автомат змінюватиме свій стан. Розглянемо послідовно дії автомата у кожному стані.

Якщо автомат перебуває у стані `A` можливі дві ситуації: 1) при опрацюванні символу `'0'` автомат перейде до стану `B` і видасть символ `'1'`; 2) при опрацюванні символу `'1'` автомат залишиться у стані `A` і видасть `'0'`.

У стані `B` при опрацюванні символу `'0'` автомат перейде до стану `A` без зміни символу, а символ `'1'` перетворить на `'0'` і перейде до стану `C`.

Стан `C` для обох можливих символів (`'0'` чи `'1'`) як результат видаватиме `'1'` і змінюватиме свій стан: для символу `'0'` автомат перейде до стану `B`, а `'1'` – до стану `A`.

Програмний код:

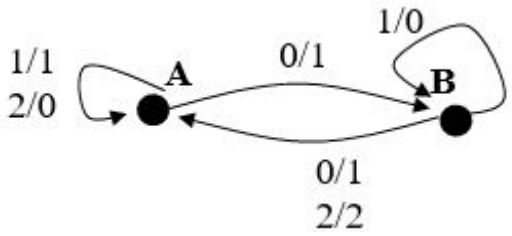
```
#include <iostream>
#include <string.h>
char* Automat1(char* S)
{
   // Для визначення станів автомата використовується перераховний тип даних
   enum conditions { A, B, C };
   conditions sit=A; // Початковий стан автомата
   for (int i=0; i<strlen(S); i++)
   switch (sit)
   {   case A: { if (S[i]=='0') {S[i]='1'; sit=B;} else S[i]='0';break;}
       case B: { if (S[i]=='0') sit=A; else sit=C; S[i]='0';break;}
       case C: { if (S[i]=='0') sit=B; else sit=A; S[i]='1';break;}
   }
   return S;
}

int main()
{
  std::cout << "Введіть вхідну послідовність для автомата (0 та 1):\n";
  char s[100];
  std::cin.getline(s,100);
  puts("Вихідна послідовність:");
  puts(Automat1(s));
}
```

Результати:

Введіть вхідну послідовність для автомата (0 та 1):
11000101011001110011

Вихідна послідовність:
00101010101100001010 1

Приклад 2. Задано автомат алфавітом X = Y = {0, 1, 2} та двома станами – A та B. Ввести вхідну послідовність символів з алфавіту X й здобути вихідну послідовність після дії заданого автомата.



Розв'язок. Цей автомат, заданий графічно, можна подати також у вигляді таблиці переходів:

Стан	Вхідний елемент		
	0	1	2
A	B,1	A,1	A,0
B	A,1	B,0	A,2

Згідно до рисунка й таблиці, якщо автомат перебуває у стані A, за елементів 1, 2 він залишиться у тому ж стані, а за елемента 0 він перейде до стану B.

Програмний код:

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

char* Automat2(char* s)
{
   enum conditions {A,B};
   conditions sit=A;
   for(int i=0;i<strlen(s);i++)
     switch (sit)
     {
        case A: { if(s[i]=='0') { sit=B; s[i]='1'; }
                  else
                     if(s[i]=='2') s[i]='0';
                  break;
                  }
        case B: { if(s[i]=='1') s[i]='0';
                  else
                  {    sit=A;
                       if(s[i]=='0') s[i]='1';
                   }
                  break;
                }
     }
```

```
    return s;
}

int main()
{
  std::cout << "Введіть вхідну послідовність для автомата (0,1 та 2):\n";
  char s[100];
  std::cin.getline(s,100);
  puts("Вихідна послідовність:");
  puts(Automat2(s));
}
```

Результати:

Введіть вхідну послідовність для автомата (0,1 та 2):
1100220111002020
Вихідна послідовність:
1111001000112121

3 Асинхронні автомати

Автомат називається *асинхронним*, якщо вхідна послідовність може бути перетворена ним на послідовність іншої довжини, тобто один символ може бути перетворено у довільну кількість символів.

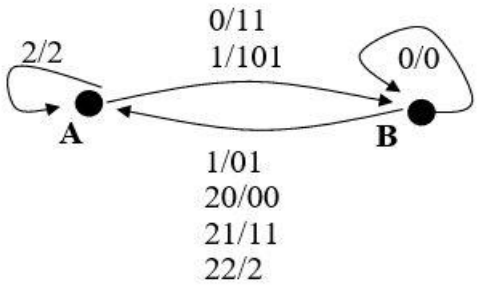
Асинхронні автомати переважно є більш складні за синхронні. Це доволі поширений вид автоматів, але надто часто асинхронні автомати можуть губити частину тексту, який вони перетворюють.

Автомат `A` називають *оборотним*, якщо існує інший автомат `B`, який перетворює довільну вихідну послідовність автомата `A` на вхідну послідовність автомата `A`. Зрозуміло, що автомат є оборотним тоді, й лише тоді, коли він не губить жодної з частин перетворюваного тексту. Вочевидь, що, якщо, приміром, автомат `A` перетворює довільний текст на порожній, то такий автомат є необоротний. Автомат неодмінно має бути оборотним, якщо його використовують для шифрування (оскільки після шифрування треба буде розшифровувати текст).

При програмуванні асинхронних автоматів для вихідного рядка доцільно використовувати нову змінну, не змінюючи вхідний рядок, як це було у разі синхронніх автоматів.

Приклади програми

Приклад 3. Задано автомат алфавітом `X=Y={0, 1, 2}` і двома станами `A` та `B`:



Ввести вхідну послідовність символів з алфавіту `X` і здобути вихідну послідовність `Y` після дії заданого автомата.

Розв'язок. Для цього автомата незавжди можливо визначити дію автомата за першим символом вхідної послідовності і виникає потреба перевірки наступного символу, тобто дія автомата визначатиметься послідовністю декількох символів. Наприклад, у стані `B` для вхідного символу `'1'` правило переходу є відоме – `1/01`, а для вхідного символу `'2'` визначити однозначно дію автомата є неможливо. Тоді для визначення правила переходу слід прочитати ще один символ вхідної послідовності (приміром, якщо другий символ є `'1'`, то перехід буде визначено: `21/11`).

Алгоритм роботи даного автомата є такий. Розглядаємо перший, а, якщо треба, то й другий символ рядка `S`, і відповідно до стану автомата використовуємо ці дані для формування рядка-відповіді `T`. Опрацьовані символи з рядка `S` вилучаємо, щоб опрацювання розпочиналося завжди лише з перших символів цього рядка і допоки рядок не стане порожнім.

Програмний код:

```
#include <iostream>
using namespace std;
#include <string.h>
void del(char *s, int n)
{
   char *s1=new char [strlen(s)+1];
   strncpy(s1, s, 0);
   s1[0]='\0';
   strcat(s1, s+n);
   strcpy(s,s1);
   delete [] s1;
}
void Assinhr1(char* S, char* T)
{
  // T- Рядок, в якому формуватиметься результат.
  enum conditions {A, B}; // Перераховний тип даних
```

```
  conditions sit=A;
  while(strlen(S)>0) // Допоки рядок не є порожній,
  {
    switch (sit) // перевіряються усі стани автомата
    {
    case A:
    {
      if(S[0]=='0') { T=strcat(T,"11"); sit=B;}
      else
         if(S[0]=='1') { T=strcat(T,"101"); sit=B;}
         else T=strcat(T,"2");
      del(S,1); // Видалення опрацьованого символу.
      break;
     } // end case A
     case B:
     {
        if (S[0]=='0') T=strcat(T,"0");
        else
        {
           if(S[0]=='1') {strcat(T,"01"); sit=A;}
           else
               if(strlen(S)>1)
                {
                   char* sub = strncpy(sub,S,2);
                   if (strncmp(sub,"20",2)==0) strcat(T,"00");
                   else
                       if (strncmp(sub,"21",2)==0) strcat(T,"11");
                       else
                          if (strncmp(sub,"22",2)==0) strcat(T,"2");
                       sit=A;
               }
      }
     if((S[0]=='2')&&(strlen(S)>=1))del(S,2);
     else del(S,1);
     break;
    } // end case B
   } // switch
  } // while
}
int main()
{
cout << "Введіть вхідну послідовність для автомата (0,1 та 2):\n";
char s1[100],s2[100];
cin.getline(s1,100);
```

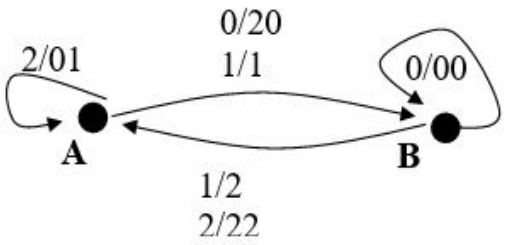
```
Assinhr1(s1,s2);
puts("Вихідна послідовність:");
puts(s2);
}
```

Результати:

Введіть вхідну послідовність для автомата (0,1 та 2):
`22211111000102002202l2`
Вихідна послідовність:
`22210101101011010000111001121111 2`

Слід зазначити, що задані в такий спосіб асинхронні автомати у певному розумінні не є строго коректними, оскільки автомат не завжди має можливість за вхідною послідовністю однозначно видати вихідну послідовність. Наприклад, як було зазначено вище, якщо в нашому автоматі вхідна послідовність є `S="02"`, то перший символ видасть вихідну послідовність `"11"`, а другий – не видасть жодного результату. Такі автомати є частково визначеними. Але вони є невизначені лише на малих "залишках" вхідної послідовності. Природно, що автомати використовують на текстах досить великої довжини. Тому невизначеність на одному-двох останніх символах не є критичною. В таких випадках автомат просто не перетворює залишок.

Приклад 4. Задано автомат алфавітом `X=Y={0,1,2}` і двома станами – `A` та `B`:



Ввести вхідну послідовність символів з алфавіту `X` і здобути вихідну послідовність після дії заданого автомата.

Розв'язок. Зрозуміло одразу, що цей автомат є асинхронним, оскільки один символ може перетворюватися на два. Але програмування його не потребує зчитування кожний раз більш ніж одного символу вхідного рядка, на відміну від попереднього прикладу. Тому зчитування символів вхідної послідовності можна організувати за допомогою циклу `for`.

У цьому разі ідея програмування автомата поєднує випадки синхронних та асинхронних автоматів.

Програмний код:

```
#include <iostream>
using namespace std;
#include <string.h>
```

```
void Assinhr2(char* S,char* T)
{
  enum conditions{A,B};
  conditions sit=A;
  for (int i=0; i<strlen(S); i++)
    switch (sit)
    {
      case A: { if (S[i]=='1') {strcat(T,"1"); sit=B;}
                 else
                    if (S[i]=='0'){strcat(T,"20"); sit=B;}
                    else strcat(T,"01");
                 break;
               } // end case A
     case B: { if(S[i]=='0') strcat(T,"00");
                 else
                   if(S[i]=='1') { strcat(T,"2"); sit=A; }
                   else  { strcat(T,"22"); sit=A; }
                 break;
             } // end case B
    } // switch
}
int main()
{
  cout << "Введіть вхідну послідовність для автомата (0,1 та 2):\n";
  char s1[100],s2[100];
  cin.getline(s1,100);
  Assinhr2(s1,s2);
  puts("Вихідна послідовність:");
  puts(s2);
}
```

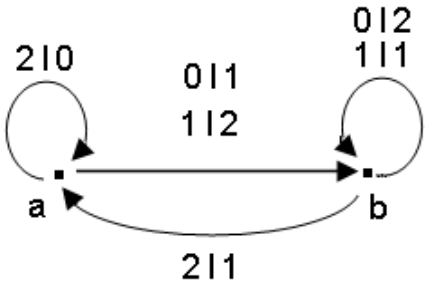
Результати роботи програми:

Введіть вхідну послідовність для автомата (0,1 та 2):
22200011110202
Вихідна послідовність:
0101012000002121002220 22

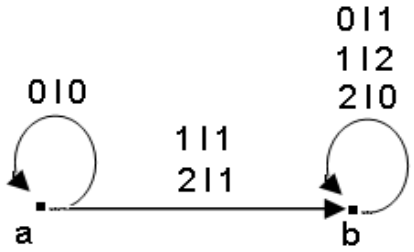
Індивідуальні завдання для самостійної роботи

Завдання 1. Синхронні автомати

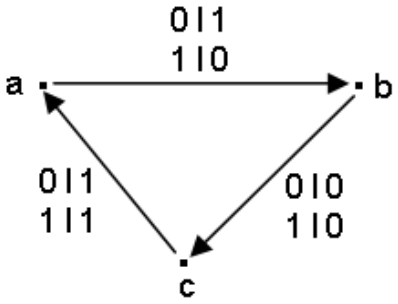
1.



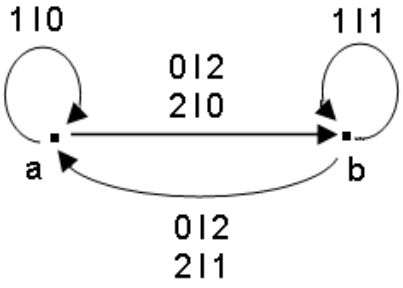
2.



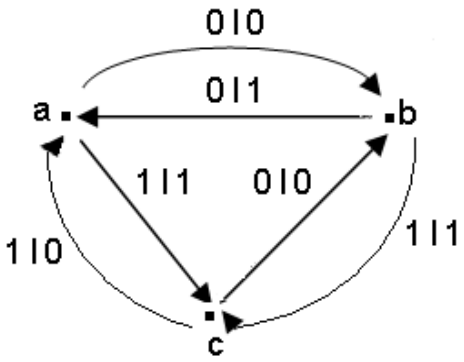
3.



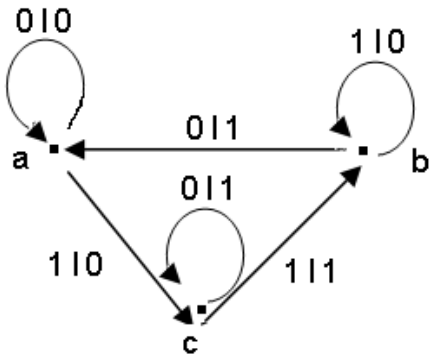
4.



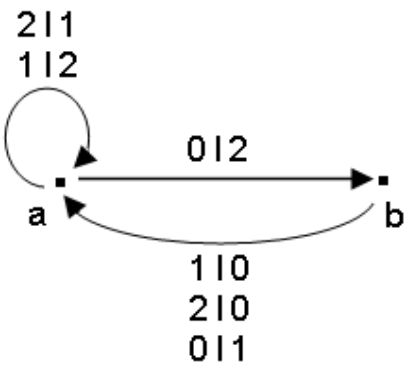
5.



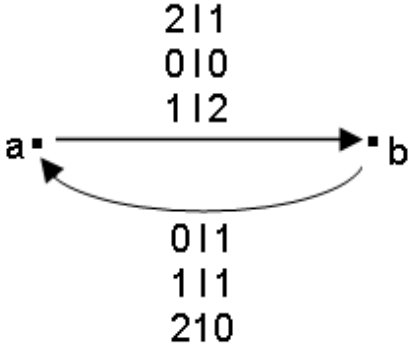
6.



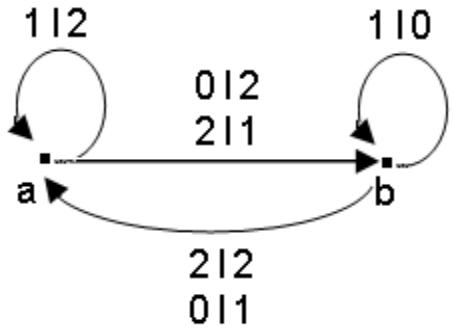
7.



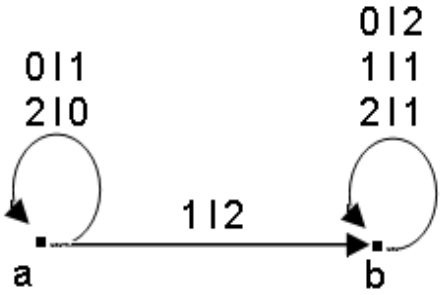
8.



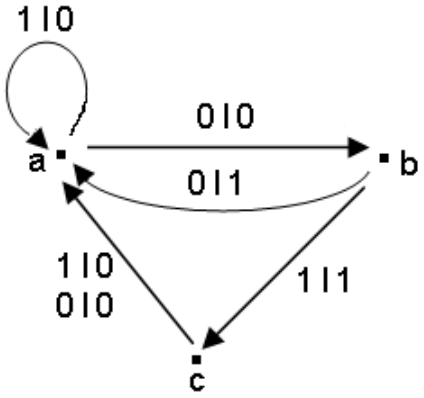
9.



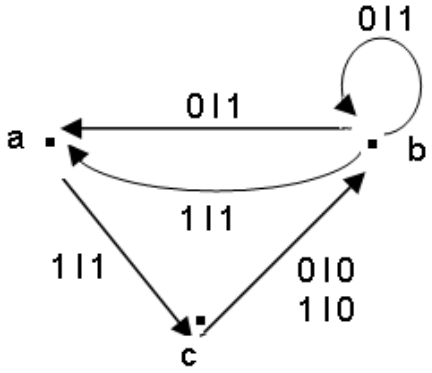
10.



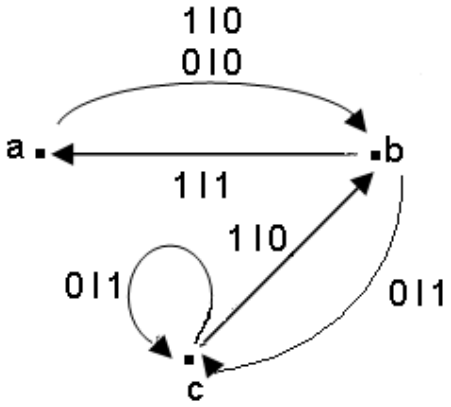
11.



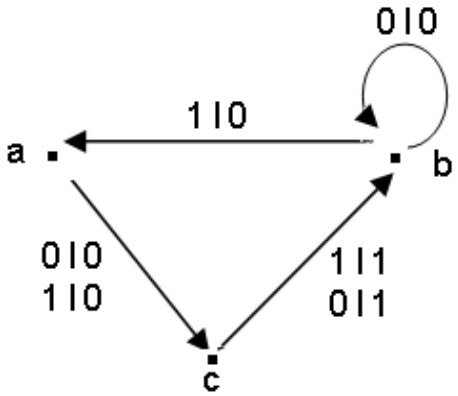
12.



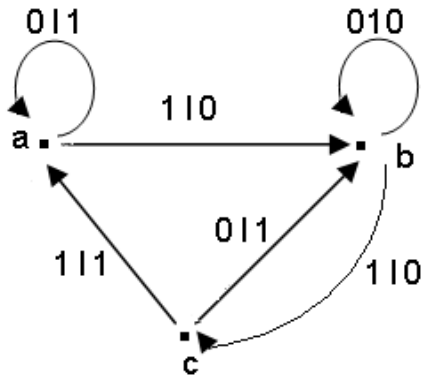
13.



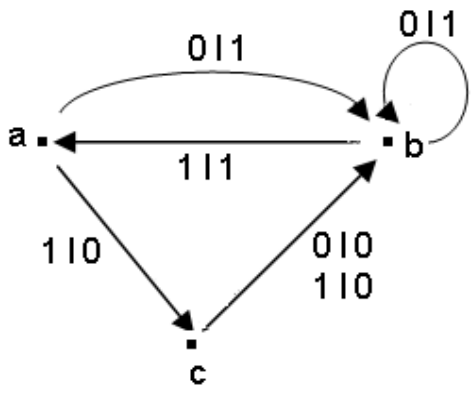
14.



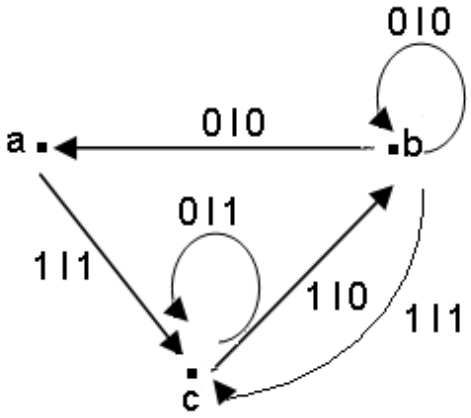
15.



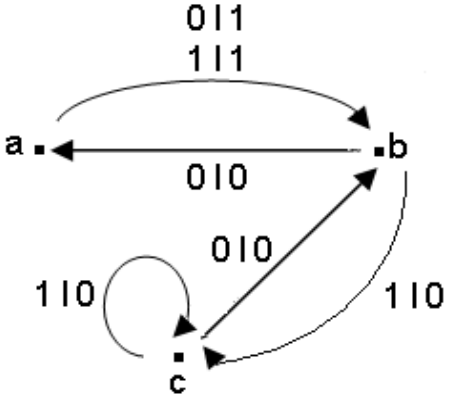
16.



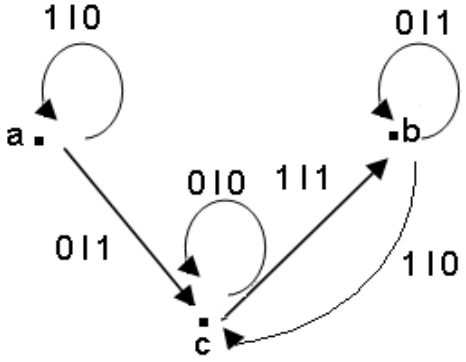
17.



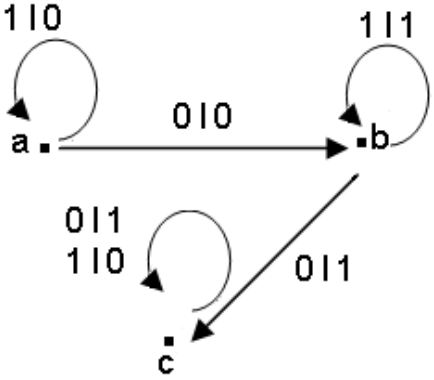
18.



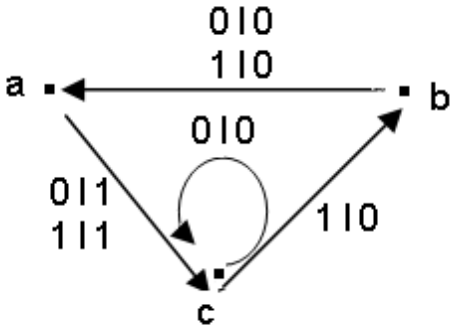
19.



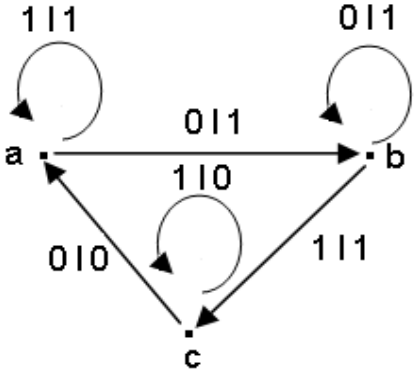
20.



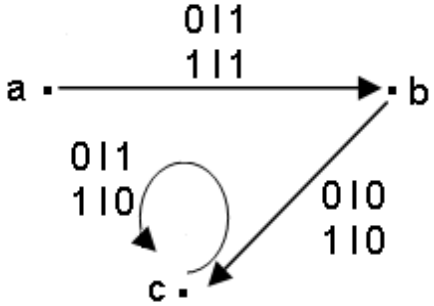
21.



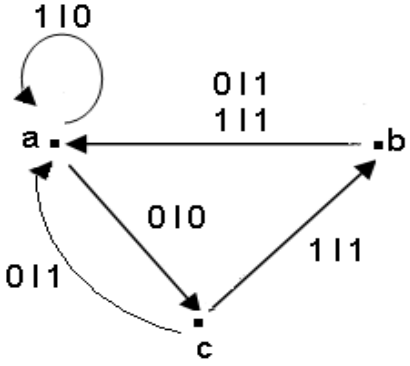
22.



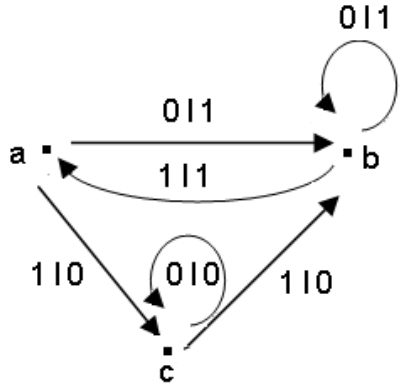
23.



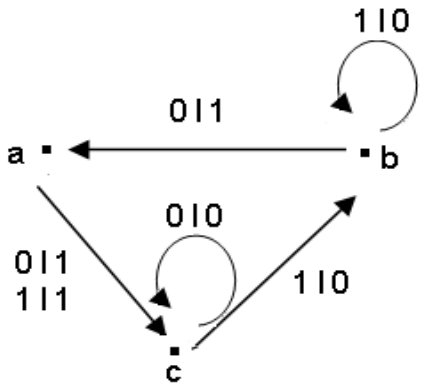
24.



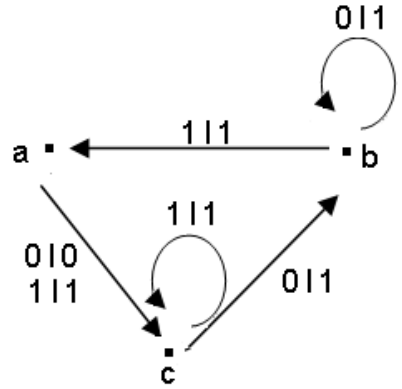
25.



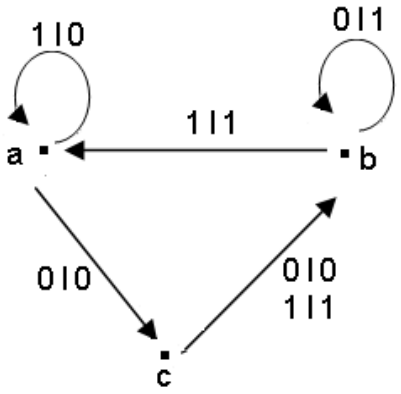
26.



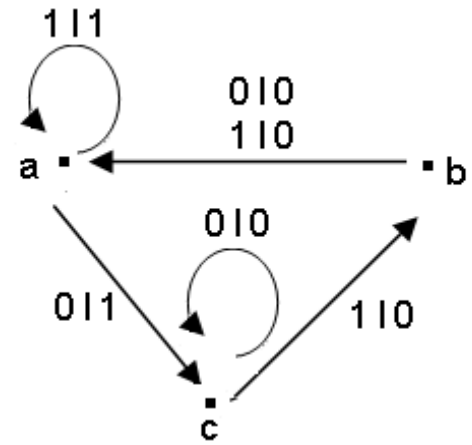
27.



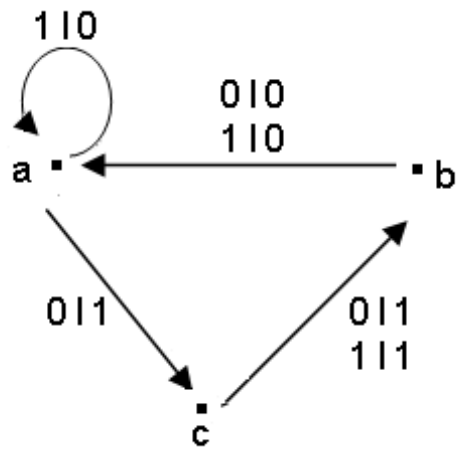
28.



29.

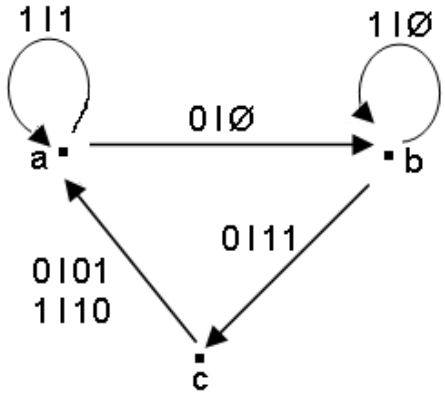


30.

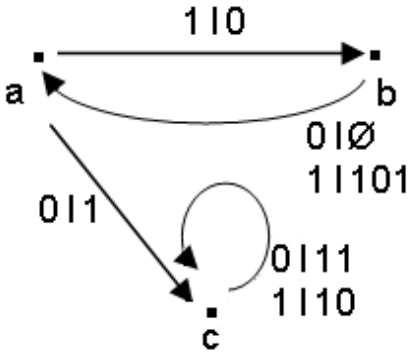


Завдання 2. Асинхронні автомати

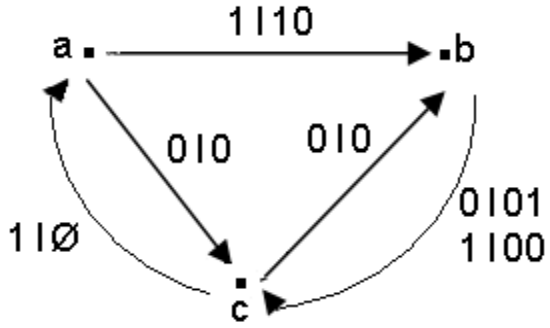
1.



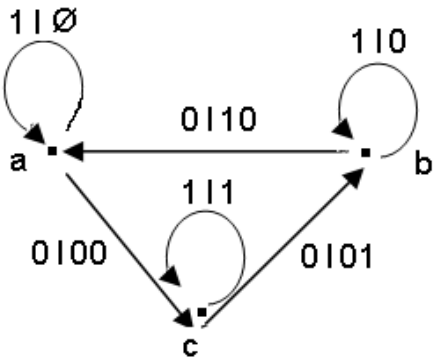
2.



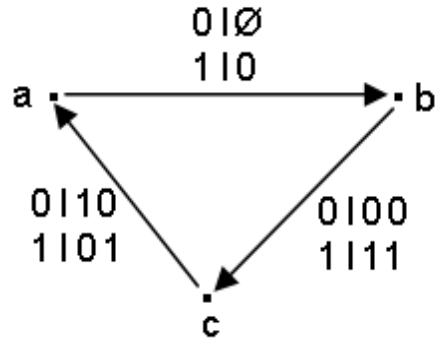
3.



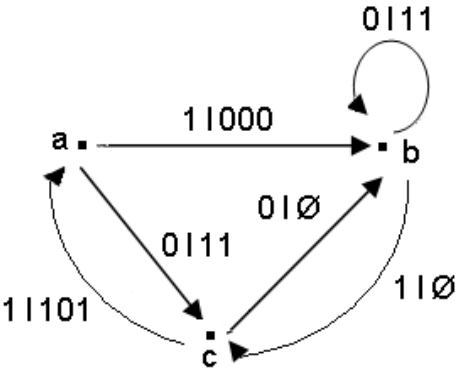
4.



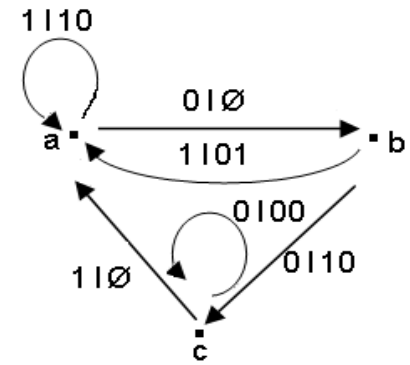
5.



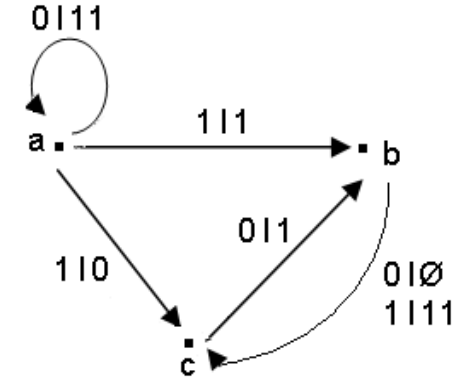
6.



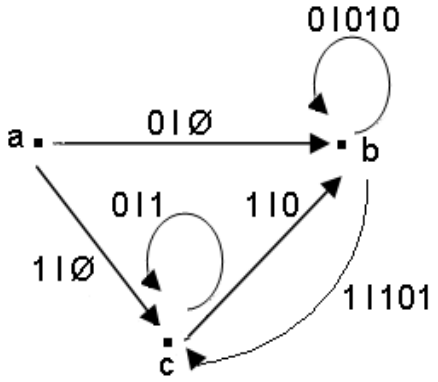
7.



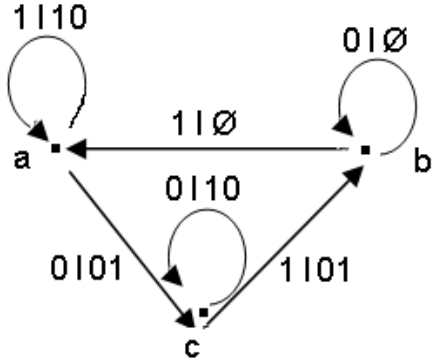
8.



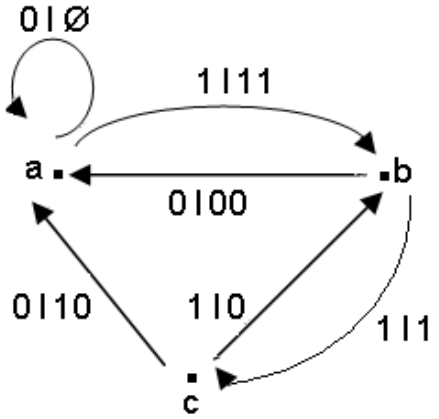
9.



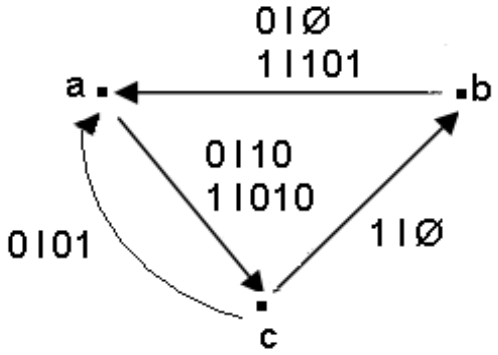
10.



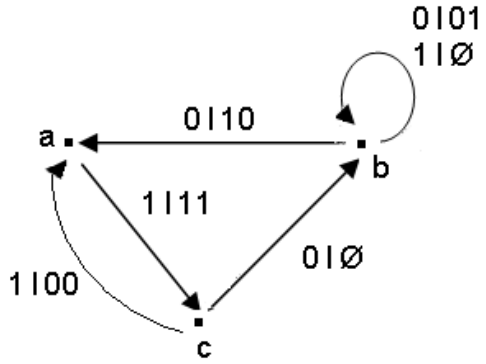
11.



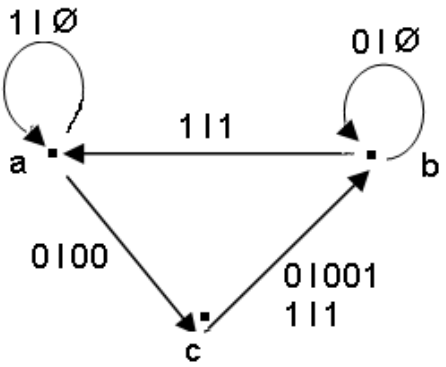
12.



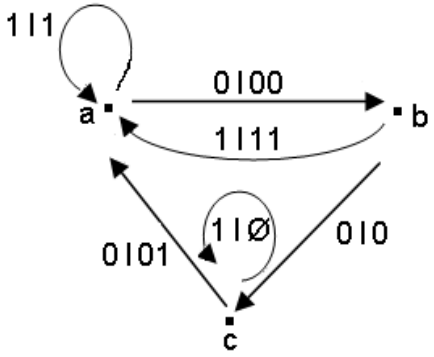
13.



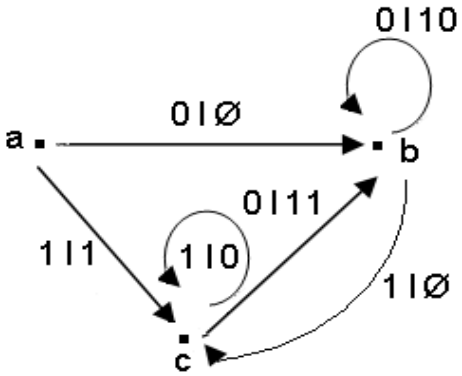
14.



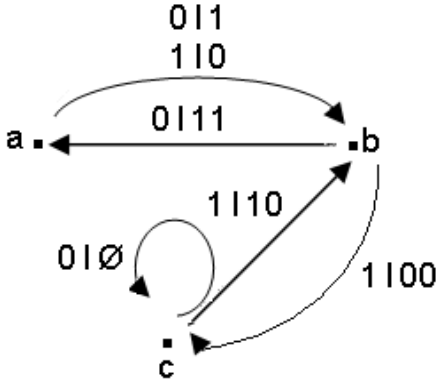
15.



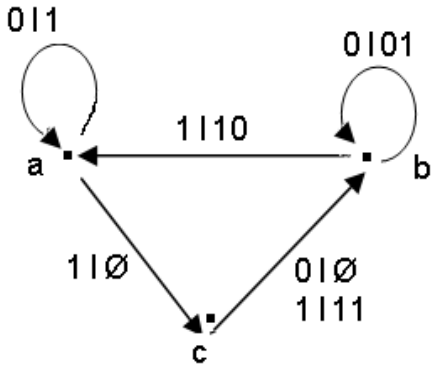
16.



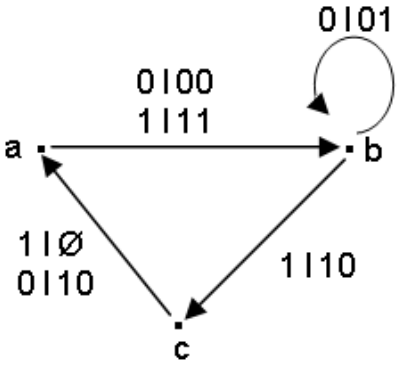
17.



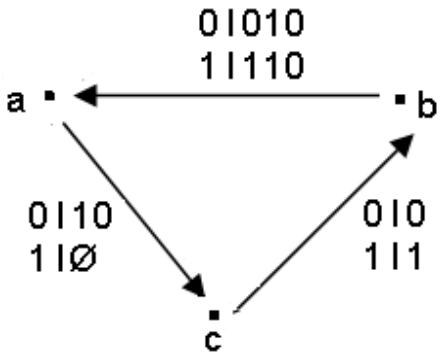
18.



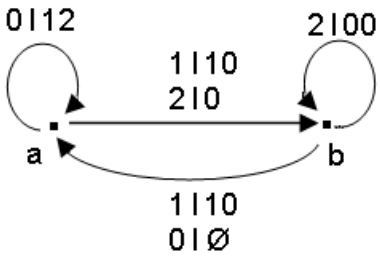
19.



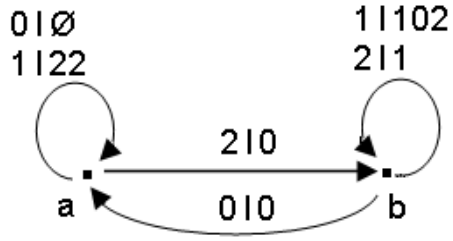
20.



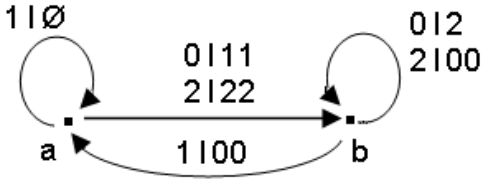
21.



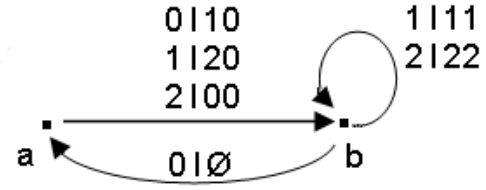
22.



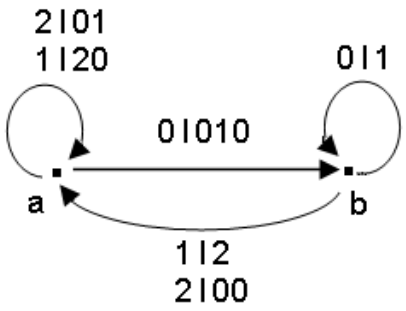
23.



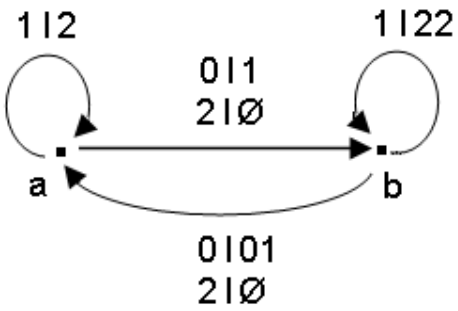
24.



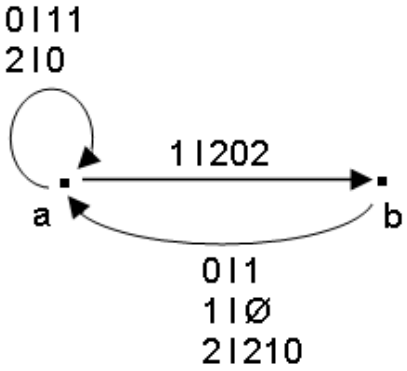
25.



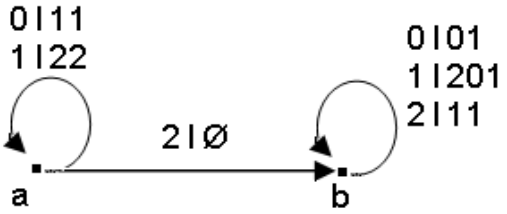
26.



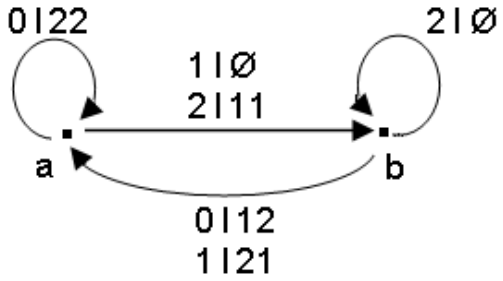
27.



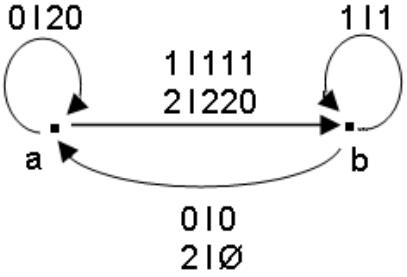
28.



29.



30.



Список літератури

1) **C++. Теорія та практика:** навч. посібник / [О. Г. Трофименко, Ю. В. Прокоп, І. Г. Швайко, Л. М. Буката та ін.] ; за ред. О. Г. Трофименко. Одеса : ВЦ ОНАЗ, 2011. 587 с.
2) **Introduction to Data Structures and Algorithms**. URL: https://www.studytonight.com/data-structures/introduction-to-data-structures.
3) **Linked List**. Data Structure and Algorithms. URL: https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm.
4) **Linked List**. URL: https://www.geeksforgeeks.org/linked-list-set-2-inserting-a-node
5) **Marcello La Rocca.** Advanced Algorithms and Data Structures. Manning Publications Co, 2021. 768 p.
6) **Mehlhorn K., Sanders P.** Algorithms and Data Structures : The Basic Toolbox/ Springer-Verlag Berlin and Heidelberg GmbH & Co, 2008. 300 p.
7) **Narasimha Karumanchi. Puzzles Taschenbuch.** Data Structures and Algorithms Made Easy: Data Structures and Algorithmic. 2016. 415 p.
8) **Алгоритми і структура даних**: навч. посібник / В. М. Ткачук. Івано-Франківськ : Видавництво Прикарпатського національного університету імені Василя Стефаника, 2016. 286 с. URL: http://194.44.152.155/elib/local/2399.pdf
9) **Алгоритми і структури даних**: опорний конспект. URL: http://dspace.wunu.edu.ua/bitstream/316497/24160/1/fkit_kn_pzs_asd_LEK.pdf
10) **Алгоритми та структури даних**: навч. посібник / Т. О. Коротєєва. Львів: Видавництво Львівської політехніки, 2014. 280 с.
11) **Алгоритми, дані і структури**: навч. посіб. / В.М. Ільман, О.П. Іванов, Л.О. Панік. Дніпропет. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. Дніпро, 2019. 134 с. URL: http://eadnurt.diit.edu.ua/bitstream/123456789/11146/1/Ilman_et_al_2019.pdf.
12) **Воробйова О.Д., Глазунова Л.В.** Алгоритми та структури даних Ч. 2. Алгоритми пошуку, стиснення даних, внутрішнього та зовнішнього сортування, алгоритми на графах : конспект лекцій. Одеса: ВЦ ОНАЗ ім. О. С. Попова, 2017. 52 с. URL : https://metod.onat.edu.ua/download/418.
13) **Воробйова О.Д., Глазунова Л.В.** Алгоритми та структури даних. Ч. 1. Структури даних : конспект лекцій. Одеса: ВЦ ОНАЗ ім. О. С. Попова, 2017. 48 с. URL : https://metod.onat.edu.ua/download/417.
14) **Мелешко Є.В., Якименко М.С., Поліщук Л.І.** Алгоритми та структури даних: навч. посібник. Кропивницький: Видавець Лисенко В.Ф. 2019. 156 с. URL: http://dspace.kntu.kr.ua/jspui/bitstream/123456789/8944/Алгоритми%20та%20структури%20даних.pdf
15) **Сборник задач по программированию**. Одесса: ОНАС им. А.С. Попова, 2011. 212 с.

Навчально-методичне видання

Олена Григорівна Трофименко,
Юлія Віталіївна Прокоп,
Олег Георгійович Янковський

Структури даних: практикум

Навчально-методичний посібник
для підготовки здобувачів вищої освіти
галузі знань 12 «Інформаційні технологій»

Електронне видання

Підписано до друку 28.06.2022.
Ум-друк. арк. 24,36. Зам. № 2206-12.

Видано в ПП «Фенікс»
(Свідоцтво суб'єкта видавничої справи ДК № 1044 від 17.09.02).
Україна, м. Одеса, 65009, вул. Зоопаркова, 25.
Тел. +38 050 7775901 +38 048 7959160
e-mail: fenix-izd@ukr.net
www.feniksbooks.com