**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**Одеська національна академія зв'язку ім. О.С.Попова**

**Кафедра інформаційних технологій**

# АЛГОРИТМИ ТА СТРУКТУРИ ДАНИХ

**Методичні вказівки до практичних та лабораторних занять з дисципліни «Алгоритми та структури даних» для студентів спеціальності 121–Інженерія програмного забезпечення та з дисципліни «Технології програмування» 125–Кібербезпека**

\

**Одеса 2020**

УДК 004.43 План НМВ 2020 р.

**Рецензент– д.т.н., доцент кафедри Інденерної механіка Військової академії (м. Одеса)– Ісмаілова Н.П.**

**Алгоритми та структури даних: методичні вказівки до практичних та лабораторних занять. / Укладач: Глазунова Л.В. – Одеса:ОНАЗ ім.О.С. Попова, 2020 - 59 с.**

Методичні вказівки містять короткі теоретичні відомості з понять і принципів створення та використання складних структур і алгоритмів. Згідно з навчальними планами дисциплін «Алгоритми та структури даних» та «Технології програмування» у методичних вказівках для закріплення теоретичного матеріалу також надані індивідуальні завдання для виконання лабораторних робіт, а також питання для самоконтролю. Для набуття практичних навичок також надено багато прикладів програм застосування складних структур та алгоритмів пошуку, сортування, пошуку найкоротших шляхів з використанням графів. Методичні вказівки будуть корисні студентам для закріплення лекційного матеріалу, а також при підготовці до практичних занять та виконанні лабораторних робіт.

Методичні вказівки призначено студентам для здобуття теоретичних і практичних знань зі спеціальності 121 – Інженерія програмного забезпечення та 125 – Кібербеспека.

СХВАЛЕНО
на засіданні кафедри
інформаційних технологій
та рекомендовано до друку.
Протокол № 6
від 27 грудня 2019 р.

ЗАТВЕРДЖЕНО
методичною радою
академії зв'язку.
Протокол № 3
від 28 січня 2020 р.

# Лабораторна робота №1.

# ОДНОСПРЯМОВАНІ ТА ДВОСПРЯМОВАНІ СПИСКИ

**Мета:** набути практичних навичок створення та застосування односпрямованих та двоспрямованих списків.

**Короткі теоретичні відомості**

***Списком*** називається впорядкована множина, що складається зі змінного числа елементів, до яких застосовані операції включення, виключення. Список, що відображає відносини сусідства між елементами, називається лінійним. ***Довжина списку*** дорівнює числу елементів, що містяться у списку, список нульової довжини називається порожнім списком. Списки є засобом організації структури даних, за якого елементи деякого типу утворюють ланцюжок. Для зв'язування елементів у списку використовують систему покажчиків.

Структура, елементами якої служать записи з одним і тим самим форматом, пов'язані один з одним за допомогою покажчиків, що зберігаються у самих елементах, називають *зв'язаним списком*. У пов'язаному списку елементи лінійно впорядковані, але порядок визначається не номерами, як у масиві, а покажчиками, що входять до складу елементів списку. Кожен список має особливий елемент, називаний покажчиком початку списку (головою списку), який зазвичай за змістом відмінний від інших елементів. В полі покажчика останнього елемента списку знаходиться спеціальна позначка NULL, яка свідчить про кінець списку.

Найбільш простою динамічною структурою є односпрямований список, елементами якого служать об'єкти структурного типу.

*Односпрямований (однозв'язний) список* − це структура даних, що являє собою послідовність елементів, в кожному з яких зберігається значення і покажчик на наступний елемент списку. В останньому елементі покажчик на наступний елемент дорівнює NULL.

Опис найпростішого елемента такого списку виглядає наступним чином:

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

де `інформаційне поле` − це поле будь-якого, раніше оголошеного або стандартного типу; `адресне поле` − це покажчик на об'єкт того ж типу, що і визначається структура, в нього записується адреса наступного елемента списку.

Наприклад:

```
struct Node {
                int key;//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
   };
```

Кожен елемент списку містить ключ, який ідентифікує цей елемент. Ключ зазвичай буває або цілим числом, або рядком. *Основними операціями, які здійснюються з односпрямованим списками, є*:

- створення списку;
- друк (перегляд) списку;
- включення елемента у список;
- виключеня елемента зі списку;
- пошук елемента у списку;
- перевірка відсутності елементів у списку (NULL);
- видалення списку.

Особливу увагу слід звернути на те, що при виконанні будь-яких операцій з лінійним односпрямованим списком необхідно забезпечувати позиціонування будь-якого покажчика на перший елемент. В іншому випадку частина або весь список буде недоступний.

Для опису алгоритмів цих основних операцій використовується наступне оголошення:

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

Для прискорення багатьох операцій доцільно застосовувати переходи між елементами списку в обох напрямках. Це реалізується за допомогою двоспрямованих списків, які є складною динамічною структурою. *Двоспрямовані (двозв'язані) списки* − це структура даних, що складається з послідовності елементів, кожен з яких містить інформаційну частину і два покажчики на сусідні елементи. При цьому два сусідні елементи повинні містити взаємні посилання один на одного.

Опис найпростішого елемента такого списку виглядає наступним чином:

```
struct ім’я_типу {
                    інформаційне поле;
                    адресне поле 1;
                    адресне поле 2;
               };
```

де `інформаційне поле` – це поле будь-якого, раніше оголошеного або стандартного, типу; `адресне поле 1` – це покажчик на об'єкт того самого типу, що і визначена структура, в нього записується адреса наступного елемента списку; `адресне поле 2` – це покажчик на об'єкт того самого типу, що і визначена структура, в нього записується адреса попереднього елемента списку. Наприклад:

```
struct list {
             type elem ;
             list *next, *prev ;
            }
list *headlist ;
```

де `type` – тип інформаційного поля елемента списку; `* Next, * prev` – покажчики на наступний і попередній елементи цієї структури відповідно. Змінна – покажчик `headlist` задає список як єдиний програмний об'єкт, її значення – покажчик на перший (або титульний) елемент списку. Основні операції, що виконуються над двоспрямованим списком, ті самі, що й для односпрямованого списку.

Розглянемо приклади створення та опрацювання односпрямованого та двоспрямованого списку.

Приклад 1.1. Створити лінійний односпрямований список з цілих чисел. Додати до списку число 11 після кожного елемента, який дорівнює 9. Результат показаний на рис. 1.1.

```
#include"iostream"
using namespace std;
//елемент односпрямованого списку
struct SingleList {
     int data;
     SingleList* next;
};
//функція створення односпрямованого списку
SingleList* Make_SingleList(SingleList* head,int n) {
     head->next = NULL;
     for (int i = 0; i <n; i++) {
          SingleList* newItem = new SingleList;
          cout << "Enter data - "<<i+1<<"=";
          cin >> newItem->data;
          newItem->next = head;
```

```
            head = newItem;
           }
      return head;
}
//функція перегляду односпрямованого списку
void Print_SingleList(SingleList* head){
      SingleList* current = head;
      while (current->next != 0)
      {    cout << current->data << "  ";
            current = current->next;
      }
      cout << endl;
}
//функція додавання елемента до односпрямованого списку
SingleList* InsertItemSingleList(SingleList* head,int DataItem){
      SingleList* Current = head;
      while(Current->next != NULL)
      {if (Current->data == 9) {
                  SingleList* NewItem = new SingleList;
                  NewItem->data = DataItem;
                  NewItem->next = Current->next;
                  Current->next = NewItem;
            }
           Current = Current->next;
       }
      return head;
}
//функція видалення односпрямованого списку
void DeleteSingleList(SingleList* head){
      if (head != NULL){
            DeleteSingleList(head->next);
            delete head;
      }
}

int main()
{
      int n;
      cout << "Enter n=";
      cin >> n;
      SingleList* head = new SingleList;// виділити пам’ять під новий
                                              елемент списку
      head = Make_SingleList(head, n);//створення односпрямованого списку
      cout << "make list=";
      Print_SingleList(head);//перегляд списку
      head = InsertItemSingleList(head,11);//додавання елелента до списку
      cout << "list with inserts=";
      Print_SingleList(head);
      cout << "Delete list" << endl;
      DeleteSingleList(head);//виделення списку
      Print_SingleList(head);
```

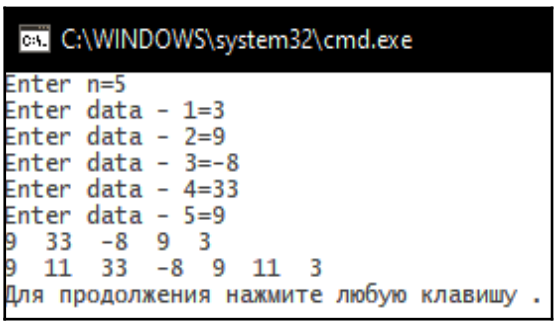



Рисунок 1.1 – Результати роботи програми прикладу 1.1

```
	return 0;
}
```

Приклад 1.2. Дано покажчик $P_1$ на перший елемент непорожнього двозв'язного списку. Видалити зі списку всі елементи з непарними значеннями та вивести покажчик на перший елемент перетвореного списку (якщо у випадку видалення елементів список стане порожнім, то вивести nul). Після видалення елементів зі списку звільнити пам'ять, яку вони займали. Результат показаний на рис. 1.2

```
#include"iostream"
using namespace std;
//елемент двоспрямованого списку
struct Double_List {
	int Data; //інформаційне поле
	Double_List *Next,*Prior; //адресне поле
};
//функція створення списку
void Make_Double_List(int n, Double_List** Head,Double_List* Prior) {
	if (n > 0) {
		//виділяємо пам'ять під новий елемент
		(*Head) = new Double_List();
		//вводимо значення інформаційного поля
		cout << "Enter data=";
		cin >> (*Head)->Data;
		(*Head)->Prior = Prior;
		(*Head)->Next = NULL;//обнулення адресного поля
		Make_Double_List(n - 1, &((*Head)->Next), (*Head));
	}
	else (*Head) = NULL;
}
// друк двоспрямованого списку
void Print_Double_List(Double_List* Head) {
	if (Head != NULL) {
		cout << Head->Data << "\t";
		Print_Double_List(Head->Next);
		//перехід до наступного елемента
	}
	//список порожній
	else cout <<"\n"<<"nul"<< "\n";
}
//видалення елемента згідно з умовою
Double_List* Delete_Item_Double_List(Double_List* Head){
	Double_List *ptr;//допоміжний покажчик
	Double_List *Current = Head;
	while (Current->Next != NULL){
		if (Current->Data % 2 != 0)
			Current = Current->Next;
		else if (Current->Prior == NULL){//видалення першого елемента
				Head = Head->Next;
				delete(Current);
```

```
                    Head->Prior = NULL;
                    Current = Head;
          }
          else
            //видалення не першого та не останнього елемента
              {     ptr = Current->Next;
                    Current->Prior->Next = Current->Next;
                    Current->Next->Prior = Current->Prior;
                    delete(Current);
                    Current = ptr;
                   }
          }
     if (Current->Next == NULL&&Current->Data%2==0) {
          //видаляємо останній елемент
          Current->Prior->Next = NULL;
          delete(Current);
          Current = Head;
     }
     return Head; }
int main()
{
     int n;
     Double_List** Head = new
Double_List*;
     Double_List* Prior = new Double_List;
     Prior = NULL;
     cout << "Enter n="; cin >> n;
     Make_Double_List(n, Head, Prior);
     Print_Double_List((*Head));
     (*Head) = Delete_Item_Double_List((*Head));
     cout << "Delete itens" << endl;
     Print_Double_List((*Head));
     return 0;
}
```

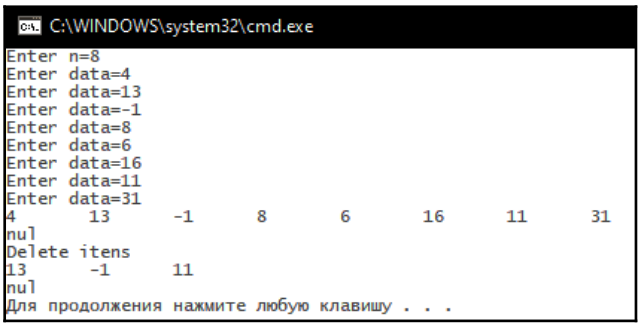



Рисунок 1.2 – Результати роботи програми прикладу 1.2

Приклад 1.3. Створити двозв'язний список груп факультету. Кожна група являє собою односпрямований список студентів. Результат показаний на рис. 1.3.

```
#include <iostream>
#include <cmath>
#include <cstring>
#include <fstream>
using namespace std;

// односпрямований список студентів
struct s
{
    char fio[15];
    s *next;
};
// двоспрямований список груп факультету
struct groups
```

```
{
    s group;
    groups *next,*prev;
};
//функція створення списку студентів
s* cr(s *begin)
{
    int n;
    cout<<"Enter count of students ";
    cin>>n;
    for(int k=1;k<=n;k++)  {
            s* i=new s;
    char str[50];
    cout<< "Enter fio "<<k<<"studenta " ;
    cin>>str;
    strcpy(i->fio,str);
    i->next=begin;
    begin=i;
    }
    return begin;
}
//функція створення групи
void v(s *begin,int k)
{
    cout<<"New list of "<<k<<"  "<<endl;
    s *i=begin;
    if (begin==NULL)
    {
        cout<<"Group is empty";
        return;
    }
    while(i!=NULL)
    {
        cout<<i->fio<<endl;
        i=i->next;
    }
    cout<< endl;
}
//функція звільнення пам’яті
void f(s *begin)
{
    s *i;
    while (begin!=NULL)
    {
        i=begin;
        begin=(begin)->next;
        delete i;
    }
    cout<<"Memory if free"<<endl;
}
//функція створення списку груп факультету
```

```
void create(groups **b, groups **e)
{
    groups *tek=new groups;
    s *b1=NULL;
    tek->group=*cr(b1);
    tek->next=NULL;
    if (*b==NULL)
    {
        tek->prev=NULL;
        *b=*e=tek;
    }
    else{
        tek->prev=*e;
        (*e)->next=tek;
        *e=tek;
    }
}

//функція перегляду списку груп факультету
void view(groups *begin){
    groups *i=begin;
    if (begin==NULL) {
        cout<<"Faculty is empty";
        return;
    }
    int k=1;
    while(i!=NULL)    {
        cout<<"Info about "<<k<<"
group:"<<endl;
        s *ind=&(i->group);
        v(ind, k);
        //cout<<i->group->fio<<endl;

        i=i->next;k++;
    }
    cout<<endl;
}
//функція звільнення пам’яті під список груп факультету
void free(groups **begin)
{
    groups *i;
    while (*begin!=NULL)     {
        i=*begin;
        s &ind=(*begin)->group;
        f(&ind);
        *begin=(*begin)->next;
        delete i;
    }
    cout<<"Memory if free"<<endl;
}
```

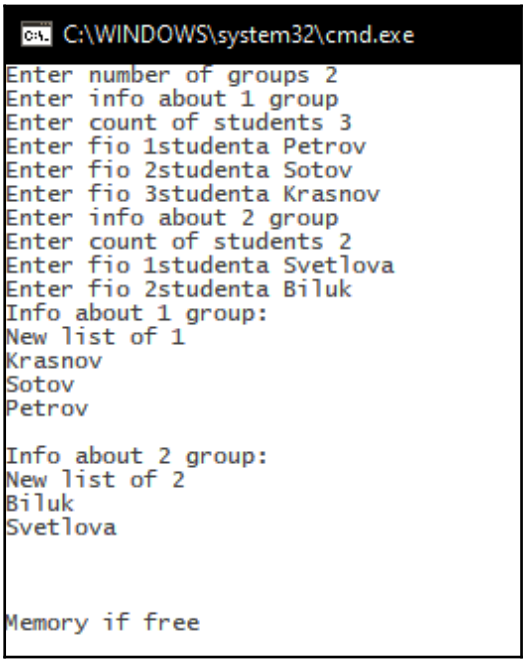



Рисунок 1.3 – Результати роботи програми прикладу 1.3

```
int main()
{
    groups *begin=NULL,*end=NULL;
    int n;
    cout<<"Enter number of groups ";
    cin>>n;
    for (int i=1;i<=n;i++) {
        cout<< "Enter info about "<<i<<" group"<<endl;
        create(&begin,&end);
    }
    view(begin);
    cout<<endl;
    free(&begin);
    return 0;
}
```

## Питання для самоконтролю

1. Чи кожен список є зв'язним? Обгрунтуйте відповідь.
2. У чому відмінність першого елемента односпрямованого (двоспрямованого) списку від інших елементів цього ж списку?
3. У чому відмінність останнього елемента односпрямованого (двоспрямованого) списку від інших елементів цього ж списку?
4. Чому при роботі з односпрямованим списком необхідне позиціонування на перший елемент списку?
5. Чому при роботі з двоспрямованим списком не обов'язкове позиціонування на перший елемент списку?
6. У чому принципові відмінності виконання додавання (видалення) елемента на першу і будь-яку іншу позиції в односпрямованому списку?
7. У чому принципові відмінності виконання основних операцій в односпрямованих і двоспрямованих списках?
8. З якою метою в програмах виконується перевірка на порожнину односпрямованого (двоспрямованого) списку?
9. З якою метою в програмах виконується видалення односпрямованого (двоспрямованого) списку після закінчення роботи з ним? Як зміниться робота програми, якщо операцію видалення списку не виконувати?

## Завдання до лабораторної роботи

1. Написати програму, у якій потрібно створити лінійний односпрямований список та виконати дії згідно з власним варіантом завдання табл. 1.1.
2. Написати програму, у якій потрібно створити двоспрямований список та виконати дії згідно з власним варіантом завдання табл 1.2.
3. Письмово відповісти на питання до лекції з односпрямованого та двоспрямованого списку.

Таблиця 1.1

| № вар. | Завдання |
|---|---|
| 1 | Створити лінійний односпрямований список із дійсних чисел. Видалити зі списку елемент перед кожним елементом, який має значення 55 |
| 2 | Створити лінійний односпрямований список із дійсних чисел. Видалити зі списку елемент після кожного елемента, який має від'ємне значення |
| 3 | Створити лінійний односпрямований список із дійсних чисел. Додати до списку число 1,5 після кожного елемента з від'ємним значенням |
| 4 | Створити лінійний односпрямований список із дійсних чисел. Визначити суму елементів списку зі значеннями більше чи рівним 15 |
| 5 | Створити лінійний односпрямований список з дійсних чисел. Видалити зі списку перший елемент менший за модулем 5 |
| 6 | Створити лінійний односпрямований список із дійсних чисел. Продублювати у списку перший додатний елемент (якщо такого немає залишити список без зміни) |
| 7 | Створити лінійний односпрямований список із цілих чисел. Видалити зі списку перший парний елемент, який знаходиться на непарній позиції |
| 8 | Створити лінійний односпрямований список із цілих чисел. Додати до списку число 66 після кожного елемента з від'ємним значенням |
| 9 | Створити лінійний односпрямований список із дійсних чисел. Видалити зі списку елемент перед кожним елементом, який має значення -2. Додати число 33 в кінець списку |
| 10 | Створити лінійний односпрямований список з цілих чисел. Видалити зі списку елемент після кожного елемента, який дорівнює 4. Додати число 0 перед кожним числом 1 |
| 11 | Створити лінійний односпрямований список із дійсних чисел. Додати до списку число 1.5 після кожного елемента з від'ємним значенням. Видалити зі списку усі числа від 2 до 5 |
| 12 | Створити лінійний односпрямований список із цілих чисел. Визначити суму елементів списку зі значенням більше або рівним 15. Видалити зі списку усі значення, які менші 5 |
| 13 | Створити лінійний односпрямований список із символів. Видалити зі списку перший елемент, код якого менше 48. Додати символ % після кожної цифри. |
| 14 | Створити лінійний односпрямований список із дійсних чисел. Додати перший додатний елемент списку після кожного від'ємного числа (якщо такого немає, залишити список без змін). |
| 15 | Створити лінійний односпрямований список із цілих чисел. Додати до списку останній парний елемент після кожного непарного елемента |
| 16 | Створити лінійний односпрямований список із цілих чисел. Додати до списку символ "!" після кожного елемента, який кратний 5 |

| № вар. | Завдання |
|---|---|
| 17 | Створити лінійний односпрямований список із дійсних чисел. Видалити зі списку елемент перед кожним елементом, який має значення в інтервалі від 10 до 20 |
| 18 | Створити лінійний односпрямований список із символів. Видалити зі списку елемент після кожного символу & |
| 19 | Створити лінійний односпрямований список із дійсних чисел. Додати до списку число 13.5 після першого елемента, який має значення більше 2 |
| 20 | Створити лінійний односпрямований список із дійсних чисел. Визначити середнє значення елементів списку, які мають значення менше або дорівнює 15. Видалити зі списку елементи, які більше 25 |
| 21 | Створити лінійний односпрямований список із цілих чисел. Видалити зі списку перший елемент більше числа 4. Додати до списку число 10 перед кожним числом, яке дорівнює 15 |
| 22 | Створити лінійний односпрямований список із дійсних чисел. Додати до списку перший від'ємний елемент перед кожним числом, яке дорівнює 20 (якщо таких немає, залишити список без змін) |
| 23 | Створити лінійний односпрямований список з цілих чисел. Видалити зі списку кожний елемент, який є кратний трьом. Додати до списку число 88 після кожної пари рівних чисел |
| 24 | Створити лінійний односпрямований список із цілих чисел. Додати до списку число 25 перед кожним елементом, який має додатне значення. Видалити зі списку усі від'ємні числа |
| 25 | Створити лінійний односпрямований список із символів. Видалити зі списку елемент перед кожним символом ^. Визначити кількість символів * у списку |
| 26 | Створити лінійний односпрямований список із дійсних чисел. Видалити зі списку елементи, у яких дробна частина більше 0,5 |
| 27 | Створити лінійний односпрямований список із цілих чисел. Додати до списку число 0 перед кожним числом від 2 до 7. Визначити суму чисел, які більше 7 |
| 28 | Створити лінійний односпрямований список із дійсних чисел. Визначити максимальне з елементів списку та включити його після кожного елемента, який має значення 1 |
| 29 | Створити лінійний односпрямований список із цілих чисел. Видалити зі списку перший елемент, який є кратним 5. Додати число 44 перед кожним числом, яке є кратним 7 |
| 30 | Створити лінійний односпрямований список з дійсних чисел. Обчислити середнє значення елементів списку та вставити число 11 перед кожним числом, яке більше середнього значення |

Таблиця 1.2

| № вар. | Завдання |
|---|---|
| | Створити двоспрямований список із дійсних чисел. Видалити зі списку елемент перед кожним елементом, який має значення 3 |
| | Створити двоспрямований список із цілих чисел. Видалити зі списку перший елемент, який має значення 10 |
| | Створити двоспрямований список із цілих чисел. Видалити зі списку останній елемент, який має значення менше 15 |
| | Створити двоспрямований список із дійсних чисел. Включити до списку число 2.5 перед кожним елементом з додатним значенням |
| | Дано покажчик $P_1$ на початок однозв'язного лінійного списку. Перетворити початковий (*однозв'язний*) ланцюжок до *двозв'язного*, в якому кожний елемент зв'язаний не тільки з наступним елементом (за допомогою поля Next), але й з попереднім (за допомогою поля Prev). Поле Prev першого елемента повинно дорівнювати nul. Вивести на екран перетворений ланцюжок у зворотному порядку |
| | Дано покажчик $P_1$ на перший елемент непорожнього двозв'язного списку. Продублювати у списку всі елементи з непарними значеннями (нові елементи додавати перед існуючими елементами з такими самими значеннями) та вивести покажчик на перший елемент перетвореного списку |
| | Дано покажчик $P_1$ на перший елемент непорожнього двозв'язного списку. Продублювати у списку всі елементи з непарними значеннями (нові елементи додавати перед існуючими елементами з такими самими значеннями) та вивести покажчик на останній елемент перетвореного списку |
| | Дано покажчик $P_1$ на перший елемент двозв'язного списку, який має не менш двох елементів. Видалити зі списку усі елементи з непарними номерами та вивести покажчик на перший елемент перетвореного списку. Після видалення елементів зі списку визволити пам'ять, яку вони займали |
| | Дано покажчик $P_1$ на перший елемент непорожнього двозв'язного списку. Видалити зі списку всі елементи з непарними значеннями та вивести покажчик на перший елемент перетвореного списку (якщо у випадку видалення елементів список стане порожнім, то вивести nul). Після видалення елементів зі списку звільнити пам'ять, яку вони займали |
| 10 | Дано число $K$ ($> 0$) та покажчик $P_0$ на один з елементів непорожнього двозв'язного списку. Перемістити у списку даний елемент на $K$ позицій вперед (якщо після даного елемента знаходиться менше $K$ елементів, то перемістити його в кінець списку). Вивести покажчики на перший та останній елементи перетвореного списку. Операції виділення та визволення пам'яті не застосовувати, поля з даними (Data) |

| № вар. | Завдання |
|---|---|
| | не змінювати |
| 11 | Дано покажчик $P_1$ на перший елемент непорожнього двозв'язного списку. Перегрупувати його елементи, перемістити всі елементи з непарними номерами в кінець списку (в тому самому порядку) та вивести покажчик на перший елемент перетвореного списку. Операції виділення та звільнення пам'яті не застосовувати, поля з даними (Data) не змінювати |
| 12 | Дано два непорожніх двозв'язних списки і зв'язані з ними покажчики: $P_A$ та $P_B$ вказують на перший та останній елементи першого списку; $P_C$ — на один з елементів другого. З'єднати початкові списки, помістив всі елементи першого списку (в тому самому порядку) після даного елемента другого списку, та вивести покажчики на перший та останній елементи об'єднаного списку. Операції виділення та звільнення пам'яті не застосовувати. |
| 13 | Дано покажчики $P_X$ та $P_Y$ на два різних елементи двозв'язного списку; елемент з адресом $P_X$ находиться у списку поперед елемента з адресом $P_Y$, але не обов'язково поряд з ним. Перемістити елементи, які знаходяться поміж даними елементами (не долучаючи дані елементи) у новий список (в тому самому порядку). Вивести покажчики на перші елементи перетвореного та нового списків. Якщо новий список стане порожнім, то пов'язаний з ним покажчик буде дорівнювати nul. Операції виділення та визволення пам'яті не застосовувати |
| 14 | Дано покажчики $P_1$ та $P_2$ на перший та останній елементи непорожнього двозв'язного списку, який містить парну кількість елементів. Перетворити список у два *циклічних двоз'язних* списки, перший з яких містять першу половину елементів початкового списку, а другий – другу половину. Вивести покажчики $P_A$ та $P_B$ на два середні елементи початкового списку (елемент з адресом $P_A$ повинен входити у перший циклічний список, а елемент з адресом $P_B$ – до другого). Операції виділення та визволення пам'яті не застосовувати |
| 15 | Дано число $K$ ($> 0$) та покажчик $P_1$ та $P_2$ на перший та останній елементи непорожнього двозв'язного списку. Здійснити *циклічний зсув* елементів списку на $K$ позицій вперед (тобто у напрямку від початку до кінця списку) та вивести покажчик на перший та останній елементи отриманого списку. Для виконання циклічного зсуву перетворення початкового списку до циклічного, після чого «розірвати» його в позиції, яка відповідає даному значенню $K$. Операції виділення та звільнення пам'яті не застосовувати |
| 16 | Дано покажчики $P_1$, $P_2$ та $P_3$ на перший, останній та поточний елементи двозв'язного списку (якщо список є порожній, то $P_1 = P_2 = P_3 =$ nul). Також дано число $N$ ($> 0$) та набір з $N$ чисел. Описати тип TList — запис з полями First, Last та Current типа PNode (поля вказують відповідно |

| | № вар. | Завдання |
|---|---|---|
| | | на *перший*, *останній та поточний* елементи списку) – та процедуру InsertLast($L$, $D$), яка додає новий елемент зі значенням $D$ у кінець списку $L$ ($L$ — вхідний та вихідний параметри типу Tlist; $D$ – вхідний параметр цілого типу). Доданий елемент стає поточним. За допомогою цієї процедури додати у кінець початкового списку даний набір чисел (у тому самому порядку) та вивести нові адреси його першого, останнього та поточного елементів |
| 17 | | Дано покажчики $P_1$, $P_2$ та $P_3$ на перший, останній та поточний елементи двозв'язного списку (якщо список є порожній, то $P_1 = P_2 = P_3 =$ nul). Також дано число $N$ ($> 0$) та набор з $N$ чисел. Використовуючи тип TList (див. завдання 16), описати процедуру InsertFirst($L$, $D$), яка додає новий елемент зі значенням $D$ на початок списку $L$ ($L$ – вхідний та вихідний параметри типу Tlist; $D$ – вхідний параметр цілого типу). Доданий елемент стає поточним. За допомогою цієї процедури додати до початку початкового списку даний набір чисел (додані числа будуть розташовується у списку в зворотньому порядку) та вивести нові адреси його першого, останнього та поточного елементів |
| 18 | | Дано непорожній двозв'язний список, перший, останній та поточний елементи, які має адреса $P_1$, $P_2$ та $P_3$. Також дано п'ять чисел. Використовуючи тип TList (див. завдання 16), описати процедуру InsertBefore($L$, $D$), яка вставляє новий елемент зі значенням $D$ перед поточним елементом списку $L$ ($L$ – вхідний та вихідний параметр типу Tlist; $D$ – вхідній параметр цілого типу). Вставлений елемент стає поточним. За допомогою цієї процедури вставити п'ять даних чисел у початковий список та вивести нові адреси його першого, останнього та поточного елементів |
| 19 | | Дано непорожній двозв'язний список, перший, останній та поточний елементи, які мають адресу $P_1$, $P_2$ та $P_3$. Також дано п'ять чисел. Використовуючи тип TList (див. завдання 16), описати процедуру InsertAfter ($L$, $D$), яка вставляє новий елемент зі значенням $D$ після поточного елемента списку $L$ ($L$– вхідний та вихідний параметр типу Tlist; $D$ – вхідний параметр цілого типу). Вставлений елемент стає поточним. За допомогою цієї процедури вставити п'ять даних чисел у початковий список та вивести нові адреси його першого, останнього та поточного елементів |
| 20 | | Створити двоспрямований список із цілих чисел. Визначити добуток парних значень елементів списку та вставити елемент зі значенням, яке дорівнює обчисленому добутку в кінець списку |
| 21 | | Створити двоспрямований список із дійсних чисел. Видалити зі списку елемент перед кожним елементом , який має значення 3 |
| 22 | | Створити двоспрямований список із цілих чисел. Видалити зі списку перший елемент , який має значення 10 |

| | № вар. | Завдання |
|---|---|---|
| 23 | | Створити двоспрямований список із цілих чисел. Видалити зі списку останній елемент, який має значення менше 15 |
| 24 | | Створити односпрямований список із дійсних чисел. Додати до списку число 2,5 перед кожним елементом з додатним значенням |
| 25 | | Створити двоспрямований список із цілих чисел. Визначити добуток парних значень елементів списку та додати елемент зі значенням, яке дорівнює обчисленому добутку в кінець списку. |
| 26 | | Дано текстовий файл. Створити двоспрямований список, кождий елемент якого містить кількість символів у відповідному рядку тексту |
| 27 | | Дано список студентів. Елемент списку містить призвище, ім’я, рік народження, курс, номер групи, оцінки з п’яти предметів. Упорядкувати студентів за курсом, причому студенти одного курсу розташовуються в алфавітному порядку. Обчислити середній бал кожної групи з кожного предмета. Визначити самого старшого студента і самого молодшого студентів. Для кожної групи знайдить найкращого з точки зору успішності студента |
| 28 | | Записи у лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двоспрямований список. Видалити елемент з заданим ключем. Додати К елементів на початок списку |
| 29 | | Записи у лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двоспрямований список. Видалити К елементів із заданими номерами. Додати К елементів на початок списку |
| 30 | | Записи у лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двоспрямований список. Видалити елемент із заданим ключем. Додати по К елементів на початок і в кінець списку |

# Лабораторна робота № 2

# СТЕК ТА ЧЕРГА.

**Мета роботи**: набути практичні навики створення та застосування структури даних стек і черга та основні принципи роботи з ними.

**Короткі теоретичні відомості**

У списках доступ до елементів відбувається за допомогою адресації, при цьому доступ до окремих елементів не обмежений. Але існують також і такі спискові структури даних, в яких є обмеження доступу до елементів. Одним з представників таких спискових структур є стековий список або просто стек.

*Стек* (англ. Stack – стопка) – це структура даних, в якій новий елемент завжди записується у її початок (вершину), і черговий елемент також завжди вибирається з її початку. У стеках використовується метод доступу до елементів LIFO (Last Input − First Output, "останнім прийшов − першим вийшов"). Найчастіше принцип роботи стека порівнюють зі стопкою тарілок: щоб взяти другу зверху, потрібно спочатку взяти верхню. Стек − це список, у якого доступний один елемент (одна позиція). Цей елемент називається вершиною стека. Взяти елемент можна тільки з вершини стека, додати елемент можна тільки на вершину стека. Наприклад, якщо записані у стек числа 1, 2, 3, то при подальшому вилучені отримаємо 3,2,1.

Стек як динамічну структуру даних легко створити на основі лінійного списку. Оскільки робота завжди йде з заголовком стека, тобто не потрібно здійснювати перегляд елементів, видалення і додавання елементів у середину або в кінець списку, то досить використовувати економічний по пам'яті лінійний односпрямований список. Для такого списку досить зберігати покажчик на вершину стека, який вказує на перший елемент списку. Якщо стек порожній, то списку не існує, і покажчик набуває значення NULL.

Наприклад, опис стека може виглядати наступним чином:

```
struct list {
             type  inf_pole;
             list *first;
            } stack;
```

Основні операції, які застосовуються до стека:

- створення стека;
- друк (перегляд) стека;
- додавання елемента на вершину стека;
- витяг елемента з вершини стека;
- перевірка порожнини стека;
- очищення стека.

*Черга* − це структура даних, що являє собою послідовність елементів, які разташовані у порядку їх надходження. Кожен новий елемент розміщується в

кінці черги; елемент, що стоїть на початку черги, вибирається з неї першим. У черзі використовується принцип доступу до елементів FIFO (First Input − First Output, "перший прийшов − першим вийшов"). У черзі доступні два елементи (дві позиції): початок черги і кінець черги. Помістити елемент можна тільки в кінець черги, а взяти елемент тільки на її початку. Прикладом може бути звичайна черга в магазині.

Чергу як динамічну структуру даних легко створити на основі лінійного списку, як односпрямованного так і двоспрямованного. Оскільки робота відбувається з обома кінцями черги, то переважно використовувати лінійний двоспрямований список, де створюються два покажчики − один на початок списку (звідки вилучаємо елементи) і один на кінець списку (куди додаємо елементи). Якщо черга порожня, то списку не існує, і покажчики набувають значення NULL.

Приклад оголошення черги за допомогою односпрямованного списку, як самий простий:

```
struct Elem{
     string gorod;   //рядок – назва міста
     int nas;        //ціле число – населення міста
     Elem *next;     //покажчик на наступний елемент черги (адреса)
};
//оголошення покажчиків на початок і кінець черги та їх початкове обнулення
Elem *first = 0, *last = 0;
```

Основні операції, які виконуються з чергою:

- створення черги;
- друк (перегляд) черги;
- додавання елемента в кінець черги;
- витяг елемента з початку черги;
- перевірка порожнини черги;
- очищення черги.

Розглянемо приклади задач з використанням стеків та черг.

Приклад 2.1. Створити стек, інформаційними полями якої є: назва книги та кількість сторінок. Додати у стек відомості про нову книгу. Організувати перегляд даних стеку та визначити кількість книжок у стеку. Результат показаний на рис. 2.1.

```
#include<iostream>
#include<string>
using namespace std;
//створення структури елемента стека
struct elem {
     string name;
     int kol;
     elem* next;
```

```
};
//покажчик на вершину стека
elem *root = 0;
//функція додавання елемента до стека
void add(int kol, string s) {
	elem* c = new elem;
	c->kol = kol;
	c->name = s;
	c->next = root;
	root = c;
}
//функція перегляду стека
void print() {
	elem* c = root;
	while (c!=0) {
		cout << c->name << "  " << c->kol << endl;
		c = c->next;
	}
}
//функція видалення стека
void del() {
	elem *c = root;
	root = root->next;
	delete c;
}
//функція обчислення кількості книжок
int book_kol() {
	int kol = 0;
	elem* c = root;
	while (c != 0)
	{ if (c->kol>10)
		kol++;
		c = c->next;
	}
	return kol;
}

int main() {
	int n, page; string name="";
	cout << "Enter n=";
	cin >> n;
	for (int i = 0; i < n;i++)
	{
		cout << "Enter name=";
		cin>>name;
		cout << "Enter kol. pages=";
		cin >> page;
		add(page, name);
	}
	print();
	int kol = book_kol();
```

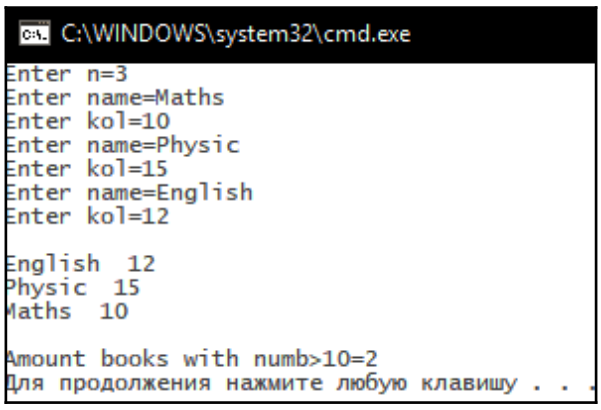



Рисунок 2.1–

```
	cout << "Sum books=" << kol << endl;
	return 0;
}
```

Приклад 2.2. Створити чергу з полями – назва міст та їх населення (у тисячах чоловік). Надати можливість користувачу додавати та видаляти елементи. Визначити місто з найбільшим населення. Видалити усі міста до міста, яке має найбільшу кількість населення. Результат показаний на рис. 2.2.

```
#include<iostream>
#include<string>
using namespace std;
//створення структури елемента черги
struct Elem{
	string gorod;	//радок – назва міста
	int nas;	//ціле число – населення міста
	Elem *next;	//покажчик на наступний елемент черги (адреса)
};
//Оголошення покажчиків на початок і кінець черги та їх початкове
обнулення
Elem *first = 0, *last = 0;

//функція додавання елемента до черги
void add(string S, int d){
	Elem *c = new Elem;
	c->gorod = S;
	c->nas = d;
	c->next = 0;
	if (first == 0)
		first = c;
	else
		last->next = c;
	last = c;
}

//Видалення першого елемента з черги
void del(){
	Elem *c = first;
	first = first->next;
	delete c;
}
//Друк черги
void print(){
	Elem *c = first;
	while (c != 0){
		cout<<c->gorod<<"  "<<c->nas<<endl;
		c = c->next;
	}
}
```

```
// Пошук міста, яке має максимальну кількість населення
void maxnas() {
      int max = first->nas;
      string maxgor = first->gorod;
      Elem *c = first;
      while (c != 0){
            if (c->nas > max)
            {
                  max = c->nas;
                  maxgor = c->gorod;
            }
            c = c->next;
      }
cout <<"City with max nas="<< maxgor << endl;
      c = first;
      while (c->gorod != maxgor){
            del();
              c = c->next;
      }
}

int main() {
      int n,nas; string gorod = "";
      cout << "Enter n="; cin >> n;
for (int i = 0; i < n;i++) {
      cout << "nas - " << i << "="; cin >> nas;
      cout << "gorod - " << i << "="; cin >> gorod;
      add(gorod, nas);
      }
cout << "\n";
print();
maxnas();
print();
return 0;
}
```

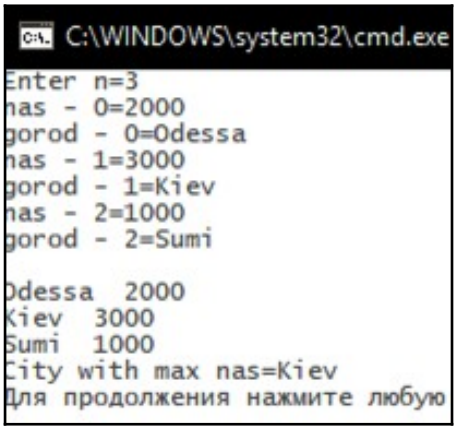



Рисунок 2.2 – Результати роботи програми прикладу 2.2

## Питання для самоконтролю

1. У чому переваги і недоліки організації структур у вигляді стека?
2. У чому переваги і недоліки організації структур у вигляді черги?
3. Для моделювання яких реальних завдань зручно використовувати стек? А для яких чергу?
4. Яке значення зберігає покажчик на стек?
5. Яке значення зберігає покажчик на чергу?
6. Які існують обмеження на тип інформаційного поля стека і черги?
7. З якою метою в програмах виконується перевірка на порожнину стека і черги?

8. При роботі зі стеком або чергою доступні позиції обмеженого числа елементів. Чи можлива ситуація запису нових елементів стека або черги на вже зайняті власними елементами ділянки пам'яті ? Відповідь обґрунтуйте.
9. З якою метою в програмах виконується видалення стека і черги після закінчення роботи з ними? Як зміниться робота програми, якщо операцію видалення не виконувати?

## Завдання до лабораторної роботи

1. Написати відповіді на питання для самоконтролю.

2. Створити динамічну структуру згідно з варіантом табл. 2.1, 2.2. У програмі повинні бути передбачені наступні функції: «Додавання елемента»; «Видалення елемента»; «Перегляд»; «Очистка». Повинні бути передбачені аварійні ситуації (наприклад: не можна видалити елемент, якщо стек порожній).

Таблиця 2.1

| № вар. | Завдання |
|---|---|
| 1 | Створити стек із цілих чисел. Обчислити добуток непарних значень елементів стека |
| 2 | Створити чергу із дійсних чисел. Визначити кількість додатних значень елементів черги |
| 3 | Дано число $N$ ($> 0$) та набір із $N$ чисел. Створити стек, який містить вхідні числа (останнє число буде вершиною стека), та вивести покажчик на його вершину |
| 4 | Створити чергу, інформаційними полями якої є: комп'ютер та обсяг його оперативної пам'яті. Видалити із черги відомості про комп'ютер, які були введені першими. Організувати перегляд даних черги й обчислити загальний обсяг пам'яті комп'ютерів, які записані у черзі |
| 5 | Дано число $D$ та покажчики $P_1$ та $P_2$ на початок та кінець черги, яка містить не менше двох елементів. Додати елемент зі значенням $D$ у кінець черги та вилучити з черги перший (початковий) елемент. Вивести значення вилученого елемента та вміст черги. Після вилучення елемента з черги звільнити пам'ять, яку займав ций елемент |
| 6 | Створити стек із дійсних чисел. Визначити максимальний елемент у стеку. Організувати перегляд даних стека |
| 7 | Створити стек, інформаційними полями якої є: прізвище та середній бал студента. Додати у стек відомості про нових студентів. Організувати перегляд даних стека. |
| 8 | Створити чергу, інформаційними полями якого є: телефон та його ціна. Видалити з черги відомості про телефон, які були введені першими. Організувати перегляд даних черги |

| № вар. | Завдання |
|---|---|
| 9 | Створити стек, інформаційними полями якого є: назва гори та висота. Додати у стек відомості про нову гору. Організувати перегляд даних стека та визначити середню висоту гір |
| 10 | Створити стек, інформаційними полями якого є: назва книги та кількість сторінок. Додати у стек відомості про нову книгу. Організувати перегляд даних стека та визначити кількість книжок устеку. |
| 11 | Створити чергу, інформаційними полями якої є: прізвище та середній бал студента. Додати у чергу відомості про нових студентів. Організувати перегляд даних черги |
| 12 | Створити чергу, інформаційними полями якої є: довжина катетів прямокутного трикутника (два дійсних числа). Додати у чергу відомості про новий трикутник. Організувати перегляд даних черги. Визначити периметр трикутника напочатку черги |
| 13 | Створити стек, інформаційними полями якого є: вулиця, номер дому та номер квартири. Додати у стек відомості про нову квартиру. Організувати перегляд даних стеку та визначити кількість домів на вулиці «Дерибасівській» |
| 14 | Дано число $N$ ($> 0$) та покажчики $P_1$ і $P_2$ на початок та кінець непорожньої черги. Вилучити з черги $N$ початкових елементів та вивести їх значення (якщо черга містить менше $N$ елементів, то вилучити усі її елементи). Після вилучення елементів з черги визволити пам’ять, яку вони займали |
| 15 | Створити чергу із дійсних чисел. Визначити мінімальний елемент черги. Організувати перегляд даних черги |
| 16 | Створити стек, інформаційними полями якого є: назва товара та його ціна. Додати у стек відомості про новий товар. Організувати перегляд даних стека та обчислити середню ціну товарів. |
| 17 | Створити чергу, інформаційними полями якої є: назва товару та його вартість. Додати у чергу відомості про новий товар. Організувати перегляд даних черги та обчислити загальну вартість товарів із назвою «Ручка шарикова» |
| 18 | Створити стек із цілих чисел. Обчислити середнє арифметичне парних значень елементів стека. Організувати перегляд даних стека. |
| 19 | Створити чергу, інформаційними полями якого є: назва процесора та його тактова частота і кількість ядер. Додати до черги відомості про новий процесор. Організувати перегляд даних черги та вивести дані про багатоядерні процесори (кількість ядер більше 1) |
| 20 | Створити чергу із цілих чисел. Визначити кількість парних значень елементів черги. Організувати перегляд даних черги |
| 21 | Створити чергу із цілих чисел. Визначити середнє значення елементів черги. Організувати перегляд даних черги |
| 22 | Створити стек, інформаційними полями якого є: назва книга та її ціна. Додати у стек відомості про нову книгу. Організувати перегляд даних |

| № вар. | Завдання |
|---|---|
| | стека та обчислити середню ціну книжок |
| 23 | Створити чергу із цілих чисел. Визначити кількість додатних елементів черги. Організувати перегляд даних черги |
| 24 | Створити чергу, інформаційними полями якої є: назва книги та її ціна. Додати до черги відомості про нову книгу. Організувати перегляд даних черги та обчислити середню ціну книжок. |
| 25 | Створити чергу із відомостей про клієнтів банку: прізвище та суми на рахунку. Визначити кількість клієнтів банку, у яких сума на рахунку понад 10000 грн. Організувати перегляд даних черги |
| 26 | Створити стек, інформаційними полями якого є: назва диску та його об'єм. Додати до стека відомості про новий диск. Організувати перегляд даних стека та визначити диск з максимальним об'ємом |
| 27 | Створити чергу з цілих чисел. Визначити кількість елементів черги менше 10. Організувати перегляд даних черги |
| 28 | Створити стек, інформаційними полями якого є: прізвище робітника та його оклад. Додати до стека відомості про нового робітника. Організувати перегляд даних стека та обчислити середній оклад |
| 29 | Створити чергу із дійсних чисел. Визначити кількість від'ємних чисел черги. Організувати перегляд даних черги |
| 30 | Створити стек, інформаційними полями якого є: назва монітора, його діагональ та його ціна. Додати до стека відомості про новий монітор. Організувати перегляд даних стека та визначити кількість моніторів з діагоналлю понад 20 дюймів. |

Таблиця 2.2

| № вар. | Завдання |
|---|---|
| 1 | Створити стек з цілими числами, використовуючи однозв'язні списки. Реалізувати операції додавання (push) та видалення (pop) елемента зі стека. Додати до стека числа 4, 3, 1, 2, 4 та роздрукувати вміст стека. Видалити один елемент зі стека, та роздрукувати вміст стека ще раз. Знайти мінімальний елемент, який належить стеку |
| 2 | Створити чергу із дійсних чисел, використовуючи однозв'язні списки. Реалізувати операції додавання (enqueue) та видалення (dequeue) елемента з черги. Додати до черги числа: -2.2, 2.3, 2.2, 5.1, 6.7 та роздрукувати вміст черги. Видалити 3 елементи з черги, потім додати до черги число 1.9 та роздрукувати чергу ще раз. Знайти додаток елементів, які належать черзі |
| 3 | Створити стек із рядків, для реалізації застосувати однозв'язні списки. Реалізувати операції додавання (push) та видалення (pop) елемента зі |

| № вар. | Завдання |
|---|---|
| | стека. Додати до стека рядки “abc”, “fx”, “glc”, “hi”, “gogo” та роздрукувати вміст стека. Видалити один елемент зі стека, потім додати рядок “the end” та роздрукувати вміст стеку ще раз. Знайти кількість рядків у стеку, які складаються з 2 символів |
| 4 | Створити чергу із рядків, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додати в чергу рядки "one", "two", "three", "four" і роздрукувати вміст черги. Видалити 2 елементи із черги, потім додати в чергу рядок "inf" і роздрукувати чергу ще раз. Знайти сумарну довжину рядків, що належать черзі, крім першого рядка черги |
| 5 | Створити стек із цілих чисел, для реалізації використовувати однозв'язані списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додати у стек числа 1, 2, 3, 4, 5 і роздрукувати вміст стека. Видалити 3 елементи зі стека, і роздрукувати вміст стека ще раз. Знайти суму елементів стека |
| 6 | Створити чергу із дійсних чисел, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента із черги. Додати в чергу числа 2.1, 2.1, 5.3 і роздрукувати вміст черги. Видалити 1 елемент із черги, потім додати в чергу число 4.9 і роздрукувати чергу ще раз. Знайти суму елементів черги |
| 7 | Створити чергу із дійсних чисел, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента із черги. Додати в чергу числа 2.2, 1.2, 2.0, 5.2 і роздрукувати вміст черги. Видалити 2 елементи із черги, потім додати в чергу число 2.9 і роздрукувати чергу ще раз. Знайти суму елементів черги |
| 8 | Створити стек із рядків, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додати у стек рядки "sdf", "2", "ssd4", "hello" і роздрукувати вміст стека. Видалити 2 елементи з стека, і роздрукувати вміст стека ще раз. Знайти рядок мінімальної довжини, що належить стеку |
| 9 | Створити чергу із рядків, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додати в чергу рядки "one", "two", "three", "four" і роздрукувати вміст черги. Видалити 1 елемент із черги, потім додати в чергу рядок "five" і роздрукувати чергу ще раз. * Знайти сумарну довжину всіх рядків, які належать черзі |
| 10 | Створити стек із цілих чисел, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додати в стек числа -5, 3, -4, 5 і роздрукувати вміст |

| № вар. | Завдання |
|---|---|
| | стека. Видалити один елемент зі стека, додати число 10 у стек, і надрукувати вміст стека ще раз. Знайти суму всіх додатних елементів, що належать стеку |
| 11 | Створити чергу із дійсних чисел, для реалізації використовувати однозв'язані списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента із черги. Додати в чергу числа 2.2, 3.2, 2.4, -3.2 і роздрукувати вміст черги. Видалити 1 елемент із черги, потім додати в чергу число 0.04 і роздрукувати чергу ще раз. Знайти суму чисел за модулем менших 1, що належать черзі |
| 12 | Створити стек із рядків, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додати у стек рядки "Students", "of", "the", "group", "TE" і роздрукувати вміст стека. Видалити один елемент зі стека, і роздрукувати вміст стека ще раз. Надрукувати всі рядки, що починаються з малої літери "t" |
| 13 | Створити чергу із рядків, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додати в чергу рядки "one", "two", "three", "four", "five", "six", "seven" і роздрукувати вміст черги. Видалити 1 елемент із черги, потім додати в чергу рядок "eight" і роздрукувати чергу ще раз. Знайти кількість рядків, що починаються з літер "s" або "t" |
| 14 | Створити стек із рядків, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додати у стек рядки "abc", "de", "f", "g", "hi", "jk" і роздрукувати вміст стека. Видалити один елемент зі стека, і роздрукувати вміст стека ще раз. Знайти кількість односимвольних рядків у стеку |
| 15 | Створити чергу із рядків, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додати в чергу рядки "one", "two", "three", "four" і роздрукувати вміст черги. Видалити 1 елемент із черги, потім додати в чергу рядок "five" і роздрукувати чергу ще раз. Знайти сумарну довжину рядків, які належать черзі |
| 16 | Створити стек із цілих чисел, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента зі стека. Додати у стек числа -5, 3, -4, 5 і роздрукувати вміст стека. Видалити один елемент зі стека, додати в стек число 10, і роздрукувати стек ще раз. Знайти суму всіх додатних елементів, які належать стеку |
| 17 | Створити чергу із дійсних чисел, для реалізації використовувати однозв'язані списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента із черги. Додати в чергу числа 46.5, 3.4, |

| № вар. | Завдання |
|---|---|
| | 32.4, -3.21 і роздрукувати вміст черги. Видалити 2 елементи із черги, потім додати в чергу число 5.0 і роздрукувати чергу ще раз. Знайти суму елементів, що за модулем більше 12 |
| 18 | Створити стек із рядків, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додати у стек рядки "Students", "of", "the", "group", "TE", "3" і роздрукувати вміст стека. Видалити один елемент зі стека, і роздрукувати вміст стека ще раз. Надрукувати всі рядки стека, які складаються з двох символів |
| 19 | Створити чергу із рядків, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додати в чергу рядки "one", "two", "three", "four", "five", "six", "seven" і роздрукувати вміст черги. Видалити 4 елементи із черги, потім додати в чергу рядки "eight", "nine" і роздрукувати чергу ще раз. Знайти кількість рядків, що складаються з 4 символів |
| 20 | Створити стек із цілих чисел, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента зі стека. Додати у стек числа 1, 4, 2 і роздрукувати вміст стека. Додайте число 4 у стек, і роздрукувати вміст стека ще раз. Знайти кількість чисел стека, які більше числа 3 |
| 21 | Створити чергу із дійсних чисел, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додати в чергу числа -2.1, 1.3, -1.34, 3.3 і роздрукувати вміст черги. Видалити 1 елемент із черги, потім додати в чергу число 2.9 і роздрукувати чергу ще раз. Знайти суму від'ємних елементів черги |
| 22 | Створити стек із рядків, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додати у стек рядки "lll", "2", "sdf4", "bye" і роздрукувати вміст стека. Видалити 1 елемент із стека, і роздрукувати вміст стека ще раз. Знайти рядок максимальної довжини, що належить стеку |
| 23 | Створити чергу із рядків, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додати в чергу рядки "one", "two", "three", "four" і роздрукувати вміст черги. Додати в чергу рядок "five" і роздрукувати чергу ще раз. Знайти сумарну довжину всіх рядків, які належать черзі, крім останнього рядка |
| 24 | Створити стек із цілих чисел, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додати у стек числа 5, 3, 44, 555 і роздрукувати вміст стека. Видалити 2 елементи зі стека, додати числа 20 і 30 у стек, і |

| № вар. | Завдання |
|---|---|
| | надрукувати вміст стека ще раз. Знайти кількість додатних двозначних чисел, що належать стеку |
| 25 | Створити чергу із дійсних чисел, для реалізації використовувати однзв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента з черги. Додати в чергу числа -2.2, 5.5, 4.3, -4.5 і роздрукувати вміст черги. Видалити 1 елемент із черги і роздрукувати чергу ще раз. Знайти суму чисел за модулем більше 4, що належать черзі |
| 26 | Створити стек із рядків, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додати у стек рядки "Student", "of", "the", "OSAT" і роздрукувати вміст стека. Видалити один елемент зі стека, додайте рядок "ONAT" і роздрукувати вміст стека ще раз. Порахувати кількість рядків, що складаються не менше ніж з трьох символів |
| 27 | Створити чергу із рядків, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента із черги. Додати в чергу рядки "one", "two", "three", "four", "five", "six", і роздрукувати вміст черги. Видалити 2 елементи із черги, потім додати в чергу рядок "seven" і роздрукувати чергу ще раз. Знайти кількість рядків, які починаються з літер "f" або "t" |
| 28 | Створити стек із цілих чисел, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (push) і видалення (pop) елемента з стека. Додати у стек числа 1, 2, -3, 4 і роздрукувати вміст стека. Видалити один елемент зі стека, і роздрукувати вміст стека ще раз. Знайти максимальний елемент, що належить стеку |
| 29 | Створити чергу із дійсних чисел, для реалізації використовувати однозв'язні списки. Реалізувати операції додавання (enqueue) і видалення (dequeue) елемента із черги. Додати в чергу числа -2.0, 2.0, 2.1, -2.1, 3.7 і роздрукувати вміст черги. Видалити 2 елементи із черги, потім додати в чергу число 1.1 і роздрукувати чергу ще раз. Знайти додаток додатних елементів, що належать черзі |
| 30 | Дано дві непорожні черги; адреси початку і кінця першої дорівнюють P1 і P2, а другий - P3 і P4. Черги містять однакову кількість елементів. Об'єднати черги в одну, в якій елементи вихідних черг чергуються (починаючи з першого елемента першої черги). Вивести покажчики на початок і кінець отриманої черги. Операції виділення і звільнення пам'яті не використовувати |

# Лабораторна робота № 3

# БІБЛІОТЕКА ШАБЛОНІВ C++ STL. ШАБЛОНИ ВЕКТОРА, СПИСКУ, СТЕКА ТА ЧЕРГИ

**Бібліотека STL** дає можливість створювати більш надійні, більш переносимі та більш універсальні програми, а також скоротити час та витрати на їх розробку.

Для роботи з динамічними структурами – послідовностями у бібліотеці **STL** використовуються наступні класи (шаблони) : vector, deque, list, stack, queue. Для роботи з цими шаблонами треба підключити бібліотеки.

#include <vector> //вставка та видалення через кінець послідовності,
#include<deque> //вставка та видалення до початку і кінця послідовності,
#include<list> //вставка та видалення до будь-якого міста послідовності,
#include <stack> // вставка та видалення через тільки початок послідовності,
#include <queue> // вставка до початку та видалення з кінця послідовності.

Операції над послідовними контейнерами

| Операція | Функція | vector | deque | list | stack | queue |
|---|---|---|---|---|---|---|
| Вставка у кінец | push_back | + | + | + | - | + |
| Видалення з кінця | pop_back | + | + | + | - | - |
| Вставка до початка | push_front | - | + | + | + | - |
| Видалення з початка | pop_front | - | + | + | + | + |
| Вставка до будь якого міста | insert | (+) | (+) | + | - | - |
| Видалення з будь якого міста | erase | (+) | (+) | + | - | - |
| Упорядкувати | sort (алгоритм) | + | + | - | - | - |

Приклад 3.1. Створити список з цілих чисел. Продемонструвати використання методів класу list. Результат показаний на рис.3.1

```
#include <iostream>
#include<list>
//#include<iterator>
using namespace std;
//функція перегляду елементів списку
void showlist(char *str, list<int> &L)
{
     list<int>::iterator i;//створення ітератору списка
     cout << str << endl;
     //L.begin()-індекс початку списку, L.end()-індекс кінця списку
     for (i = L.begin(); i != L.end(); i++)
```

```
            cout << *i << " ";
      cout << endl;
}

int main()
{
      list<int> L; //оголошення списку
      int x;
      cout << "Enter integers:\n";
      while (cin >> x, x != 0)
            L.push_back(x);//ініціалізація списку
      showlist("Initial list", L);
      L.push_front(123);//додавання елемента на початок списку
      showlist("After insert 123 at in front:", L);
       //створення та ініціалізація індекса списку
      list<int>::iterator i = L.begin();
      L.insert(++i, 456);// вставка елемента 456 у другу позицію
      showlist("After insert 456 at second position:", L);
      i = L.end();
      L.insert(--i, 999);
      showlist("After insert 999 just before end:", L);
      i = L.begin(); x = *i;
      L.pop_front();
      cout << "Delete at front: " << x << endl;
      showlist("After deletion: ", L);
      i = L.begin();
      x = *i++; cout << "To be deleted: " << x << endl;
      L.erase(i); //видалення елемента по його індексу
      showlist("After this deletion: ", L);
      return 0;
}
```

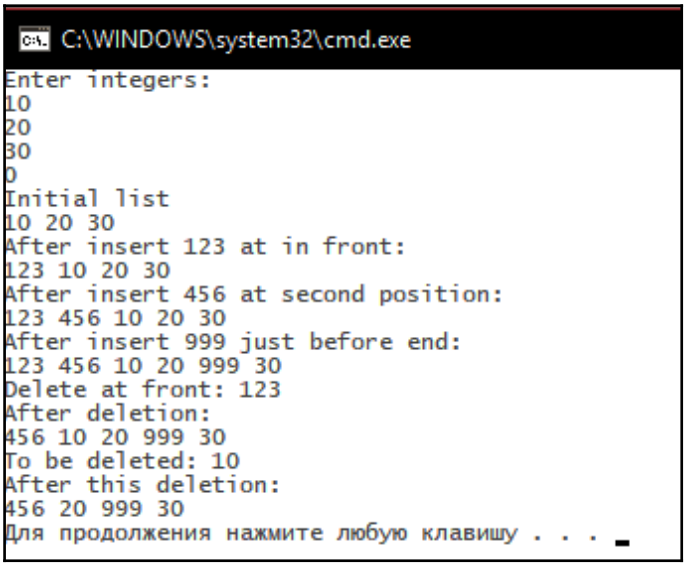



Рисунок 3.1 –

Приклад 3.2. Створити стек. Продемонструвати роботу методів класу stack. Результат показаний на рис. 3.2.

```
#include <iostream>
#include <stack>  // підключити бібліотеку для  використання стека
using namespace std;

int main() {
setlocale(LC_ALL,"rus");
stack <int> steck;  // створюємо екземпляр стека
   int i = 0;
   cout << " Enter any six integers: " << endl;

  while (i != 10) {
    int a;
    cin >> a;
    steck.push(a);  // додавання елементів у список
    i++;
        }
 // перевіряємо стек на наявність елементів
 if (steck.empty()) cout << "Stack don't empty";
 // виводимо перший елемент
cout << "First stack item: " << steck.top() << endl;
cout << "Delete first item: " << endl;
steck.pop();  // видалити перший елемент
// виводимо вже новий  перший елемент
cout << "New first item: " << steck.top();
system("pause");
return 0;
}
```

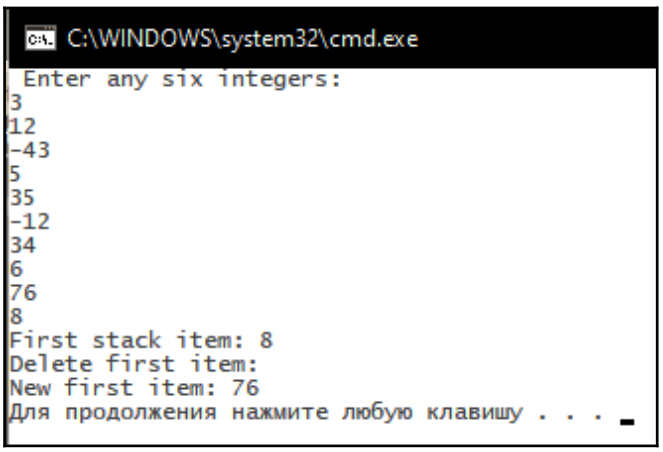



Рисунок 3.2 –

Приклад 3.3. Створити чергу. Продемонструвати роботу методів класу queue. Результат показаний на рис. 3.3.

```
#include <iostream>
#include <queue>  // підключення бібліотеки queue
using namespace std;
int main() {
  setlocale(LC_ALL,"rus");
  queue <int> q;  // створити чергу q
  cout << "Введіть 7 чисел: " << endl;
```

```
  for (int h = 0; h < 7; h++) {
    int a;
    cin >> a;
    q.push(a);  // додаємо у чергу елементи
  }
  cout << endl;
  // виводимо перший елемент
  cout << "Перший елемент у черзі: " << q.front()<< endl;
  q.pop();  // видалити елемент з черги
  cout << "Новий перший елемент (після видалення): " << q.front() <<
endl;
  // перевіряємо порожнину черги
  if (!q.empty()) cout << "Черга не порожня!";    system("pause");
  return 0;
}
```

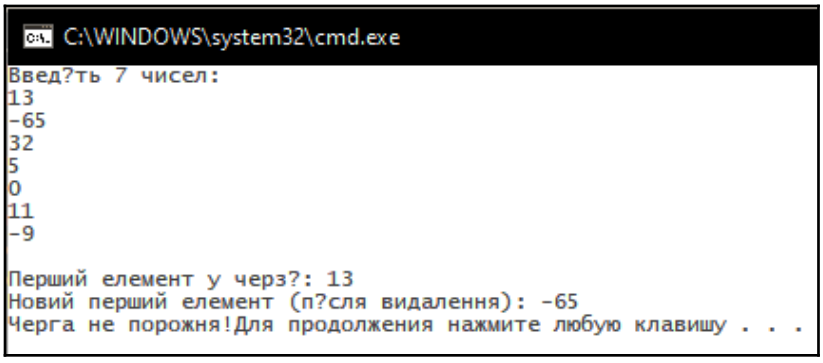



Рисунок 3.3 – Результати роботи
програми прикладу 3.3

## Питання для самоконтролю

1. Що таке шаблонні функції та класи у C++. Дати приклади створення шаблона функції та класу.
2. Для чого використовується бібліотека STL? Як підключити шаблони до проекту?
3. Коли застосовується шаблон `vector`? Основні функції роботи з шаблоном `vector`.
4. Які операції підтримує шаблон `list`? Записати функції, які їх реалізують.
5. Які операції підтримує шаблон `stack`? Запишить функції, які їх реалізують.
6. Які операції підтримує шаблон `queue`? Запишить функції, які їх реалізують.

## Завдання до лабораторної роботи

1. Написати відповіді на питання для самоконтролю.

2. Створити список та виконати завдання згідно зі своїм варіантом табл. 1.1, 1.2 з використанням шаблонів бібліотеки STL.

3. Створити стек чи чергу згідно зі своїм варіантом табл. 2.1 з використанням шаблонів бібліотеки STL.

## Лабораторна робота № 4

## ДВІЙКОВІ ДЕРЕВА ПОШУКУ (BTS).

**Мета:** набути практичних навичок створення та використання двійкового дерева пошуку, як динамічної структур даних.

Найбільш часто у задачах пошуку та сортування використоються двійкові дерева пошуку.

*Двійкове (бінарне) дерево пошуку* (англ. binary search tree, BST) — це двійкове дерево, для якого виконуються наступні додаткові умови (властивості дерева пошуку):

- обидва піддерева , ліве та праве, є двійковими деревами пошуку;
- у всіх вузлах лівого піддерева довільного вузла X значення ключів (інформаційне поле вузла) меньше, ніж значення ключа даних самого вузла X. В усіх вузлах правого піддерева довільного вузла X значення ключів даних більші або дорівнюють значенням ключа даних самого вузла X.

Ліпший засіб обходу дерева пошуку – симетричний, він дозволяє отримати впорядковану послідовність ключів. Очевидно, що для побудови двійкового дерева пошуку необхідно порівнювати значення ключів вузла, лівого та правого нащадка. Для будь–якого вузла X виконюються властивості дерева пошуку: *key*[*left*[X]] < *key*[X] ≤ *key*[*right*[X]], тобто ключі даних батьківского вузла більше ключів даних лівого нащадка та нестрого менше ключів даних правого нащадка.

Двійкове дерево пошуку не слід плутати з двійковою купою, яка побудована по іншими правилами. Основною перевагою двійкового дерева пошуку перед іншими структурами даних є можлива висока ефективність реалізаціїї заснованих на неї алгоритмів пошуку та впорядкування.

*Двійкове дерево пошуку застосовується для побудови більш абстрактних структур, таких, як множина, мультимножина, асоціативні масиви.*

Дамо простий приклад створення та використання BTS.

Приклад 4.1. З вихідної послідовності цілих чисел побудувати двійкове дерево пошуку. Обчислити середнє арифметичне значення усіх вузлів і знайти мінімальний елемент. Результати показані на рис. 4.1.

Функція *First*() на виході отримує перший елемент дерева і створює кореневу вершину з нульовими значеннями для правого і лівого нащадка. Функція *Search_Insert*() на виході отримує покажчик на корень дерева та елемент для пошуку або , якщо такого елемента немає, додаємо його у список за правилами створення двійкового дерева пошуку.

```
#include<stdio.h>
```

```
#include<stdlib.h>

struct BinaryTree {
      int data;
      BinaryTree *left, *right;
};
// створення першого вузла бінарного дерева
BinaryTree *First(int x) {
      BinaryTree *pv = new BinaryTree;
      pv->data = x;
      pv->left = 0;
      pv->right = 0;
      return pv;
}
//пошук ключа x , якщо такого ключа немає, то вставляємо
BinaryTree *Search_Insert(BinaryTree *q, int x) {
      BinaryTree *pv = q, *prev;
      bool found = false;
      while (pv&&!found)
      {
            prev = pv;
            if (x == pv->data) found = true;
            else if (x < pv->data)
                  pv = pv->left;
                else pv = pv->right;
      }
      if (found)  return pv; // якщо ключ x є у дереві, то
                                    повертається його адреса
      BinaryTree *pnew = new BinaryTree;// додаємо нового ключа
      pnew->data = x;
      pnew->left = 0;
      pnew->right = 0;
      if (x < prev->data) prev->left = pnew; //якщо x менше попередьої
      else prev->right = pnew;  //якщо x більше попередьої вершини
      return pnew;  //повертаємо адресу на новий ключ дерева
}
//перегляд бінарного дерева
void Print_BinaryTree(BinaryTree  *q, int n) {
      if (q)
      {
            int i;
            Print_BinaryTree(q->left, n+5);
            for (i = 0; i < n; i++) // для уявлення дерева
                  printf(" ");
            printf("%d", q->data);
            Print_BinaryTree(q->right, n+5);
      }
}
//видалення бінарного дерева
void Delete_BinaryTree(BinaryTree  *q) {
      if (q != NULL)
```

```
	{
		Delete_BinaryTree(q->left);
		Delete_BinaryTree(q->right);
		delete q;
	}
}
// обчислення суми значень вершин дерева
int sum_BinaryTree(BinaryTree  *q) {
	int l, r;
	if (q != NULL)
	{  //коротка форма запису оператора if
		l = (q->left != NULL) ? sum_BinaryTree(q->left) : 0;
		r = (q->right != NULL) ? sum_BinaryTree(q->right) : 0;
		return l + r + q->data;
	}
	return 0;
}
//пошук мінімального елемента двійкового
дерева пошуку
BinaryTree * min_BinaryTree(BinaryTree  *q)
{

	if (q->left == NULL) return q;
	return min_BinaryTree(q->left);

}

int main()
{
	int i,n=0;
	//об’являємо та ініціалізуємо масив цілих чисел
	int a[10] = { 1, -2, 3, 4, 15, 6, -7, 8, 0, 10 };
	//створюємо двійкове дерево пошуку
	BinaryTree  *root = NULL;
	for (i = 0; i < 10;i++)
	  if(root==0) root=First(a[i]);
	  else Search_Insert(root, a[i]);
	printf("Binary Tree=\n");
	Print_BinaryTree(root, n);
	printf("count level=%d \n",n);

	printf("avg= %5.2f\n", (float)sum_BinaryTree(root) / 10);
	printf("\n");
	BinaryTree *x = min_BinaryTree(root);
	printf("min= %d\n", x->data);
	Delete_BinaryTree(root);

	return 0;
}
```

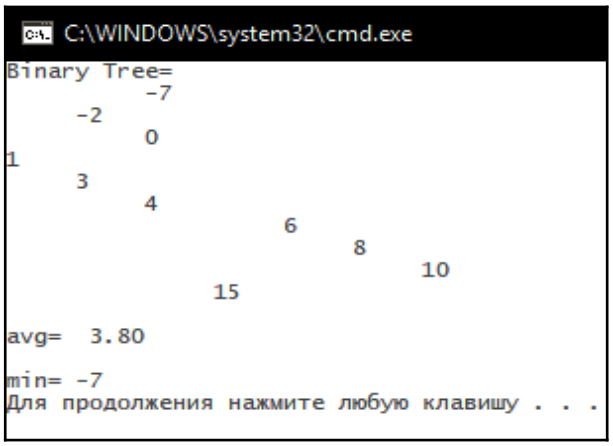



Рисунок 4.1 – Результати роботи програми прикладу 4.1

**Шаблони STL, які використовують динамічні структури – сбалансовані двійкові дерева: *map* та *set*.**

Шаблон ***map*** (***include map***) застосовується для зберігання та вилучення даних з колекції, в якій кожний елемент є пара (*k*,*d*), що одночасно має значення даних (*d*) і ключа сортування (*k*). Значення ключа унікальне і застосовується для автоматичного сортування даних. Значения даних можливо змінювати напряму. Значення ключа є константою, і його неможно змінювати. Замість цього значення ключів, які зв'язані зі старими даними, необхідно видалити та вставити нові значення ключів для нових даних. Разом з шаблоном ***map*** також застосовується шаблон ***pair*** (пара), який надає більше можливостей. Основні методи шаблана map: ***insert()*** вставка елемента у колекцію, ***erase()*** видалення елемента з колекції, ***at()*** та ***find()*** пошук даних за ключем, ***end()*** повернення ітератора після кінця, ***empty()*** перевірка на відсутність елементів у колекціх та інші.

Якщо потрібна колекція з однаковими ключами можливо застосувати контейнер ***multimap***.

Приклад 4.2 Створити та застосувати колекцію ***map*** – каталог письменників та кількість їх творів. Результати показані на рис. 4.2.

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

int main()
{   // об'ява та ініціалізація колекції map
     map <string, int> m1 = { { "Tolstoy", 36 }, { "Bulgakov", 9 } };
     map <string, int> ::iterator m1_Iter;// ітератор односпрямований
     typedef pair <string, int> Int_Pair; // об'ява колекції pair

     //додавання пар до колекції за допомогою шаблону pair
     m1.insert(Int_Pair("Green", 10));
     m1.insert(Int_Pair("Dikens", 21));
     m1.insert(Int_Pair("Semenov", 16));
     m1.insert(Int_Pair("London", 45));
     m1.insert(Int_Pair("Richardson",11));
     //переглянути колекцію
     for (m1_Iter = m1.begin(); m1_Iter != m1.end(); ++m1_Iter)
     {
          cout << m1_Iter->first << " : " << m1_Iter->second << endl;
     }
    m1_Iter = m1.begin();
     // переглянути перший елемент колекції
     cout << "The first element of m1 is " << m1_Iter->first << endl;
     // видалити перший елемент колекції
     m1.erase(m1_Iter);
     m1_Iter = m1.begin();
     // переглянути перший елемент колекції
     cout << "The first element of m1 is now " << m1_Iter->first <<
endl;
```

```
      //визначити розмір колекції
      int i = m1.size();
      cout << "The size of the map is initially " << i << "." << endl;
      //знайти кількість книг у письменика
"London"
      cout <<"Count books "<<
m1.at("London") << endl;
      // очистити колекцію
      m1.clear();
      i = m1.size();
      cout << "The size of the map after
clearing is "<< i << "." << endl;
      return 0;
      }
```

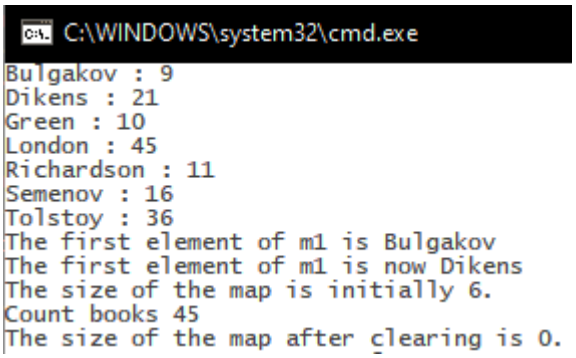



Рисунок 4.2 – Результати роботи програми прикладу 4.2

***Set*** – це асоціативний контейнер, який підтримує унікальні ключи та забеспечує швидкий пошук ключів. Контейнер ***set (#include set***) реалізує такі об'єкти як множина та мультімножина. Це є контейнери, які містять деяку кількість впорядкованих елементів. При додаванні нового елементу в множину, він одразу стає на своє місце так, що би не порушувати порядок впорядкування, тому що в множині та мультімножині всі елементи впорядковуються автоматично. Множина містить тільки унікальні елементи, а мультімножина може містити дублікати.

## Питання для самоконтролю

1. З чим пов'язана популярність використання дерев у програмуванні?
2. Чи можна список віднести до дерев? Відповідь обґрунтуйте.
3. Які дані містять адресні поля елемента бінарного дерева?
4. Чи може бінарне дерево бути чітке і неповним? Відповідь обґрунтуйте.
5. Чи може бінарне дерево бути нечітке і повним? Відповідь обґрунтуйте.
6. Куди може бути доданий елемент у збалансоване бінарне дерево? Вид дерева при цьому повинен зберегтися.
7. Чим відрізняються, з точки зору реалізації алгоритму, прямий, симетричний і зворотний обходи бінарного дерева?

## Завдання до лабораторної роботи

1. Надати відповіді на питання для самоконтролю.
2. Створити динамичну структуру бінарного дерева пошуку згідно з власним варіантом табл. 4.1. У програмі повинні бути наступні функції: «Створення структури», «Додати елемент»; «Видалити елемент»; «Перегляд».
3. Створити колекцію ***map*** для парних варіантів – довідник лік (назва, призначення), для непарних – каталог «предметний вказівник» ( термін, номер сторінки). Продемонструвати основні функції шаблону ***map.***

Таблиця 4.1

| № вар. | Завдання |
|---|---|
| 1 | З вихідної послідовності дійсних чисел побудувати бінарне дерево пошуку. Визначити рівень вузла з мінімальним елементом |
| 2 | З вихідної послідовності дійсних чисел побудувати бінарне дерево пошуку. Вивести на екран значення у вузлах, які знаходяться на шляху від кореня до вузла з максимальним елементом |
| 3 | З вихідної послідовності цілих чисел побудувати бінарне дерево пошуку. Визначити рівень вузла, який містить число з мінімальним значенням |
| 4 | З вихідної послідовності дійсних чисел побудувати бінарне дерево пошуку та визначити його глибину |
| 5 | З вихідної послідовності дійсних чисел побудувати бінарне дерево пошуку. Вивести на екран значення в тих вузлах, які мають рівень, заданий користувачем |
| 6 | З вихідної послідовності цілих чисел побудувати бінарне дерево пошуку. Вивести на екран значення всього листя дерева |
| 7 | З вихідної послідовності дійсних чисел побудувати бінарне дерево пошуку. Визначити середнє арифметичне значення у тих вузлах, які мають заданий користувачем рівень |
| 8 | З вихідної послідовності цілих чисел побудувати бінарне дерево пошуку. Визначити суму значень того листя дерева, які мають індекс парний |
| 9 | З вихідної послідовності цілих чисел побудувати бінарне дерево пошуку. Визначити суму значень того листя дерева, які мають індекс непарний |
| 10 | З вихідної послідовності цілих чисел побудувати бінарне дерево пошуку. Визначити середнє арифметичне значення вузлів з парними значеннями |
| 11 | З вихідної послідовності дійсних чисел побудувати бінарне дерево пошуку. Для кожного рівня вивести на екран значення вузлів, які знаходяться на цьому рівні зліва направо |
| 12 | З вихідної послідовності цілих чисел побудувати бінарне дерево пошуку. Вивести на екран значення усього листя дерева з непарними значеннями |
| 13 | З вихідної послідовності дійсних чисел побудувати бінарне дерево пошуку та визначити кількість вузлів, у яких є обидва ненульові нащадки |
| 14 | З вихідної послідовності цілих чисел побудувати бінарне дерево пошуку. Визначити кількість вузлів зі значеннями у діапазоні від 3 до 9 |
| 15 | З вихідної послідовності дійсних чисел побудувати бінарне дерево пошуку та визначити кількість листя дерева, які є правими нащадками вершин |
| 16 | З вихідної послідовності цілих чисел побудувати бінарне дерево пошуку. |

| № вар. | Завдання |
|---|---|
| | Для кожного з рівнів даного дерева, починаючи з нульового, вивести кількість вузлів, які знаходяться на цьому рівні |
| 17 | З вихідної послідовності дійсних чисел побудувати бінарне дерево пошуку. Вивести максимальне зі значень його внутрішніх вершин (тобто вершин, які не є листя) |
| 18 | З вихідної послідовності цілих чисел побудувати бінарне дерево пошуку. Вивести мінімальне зі значень його вузлів, які є листя |
| 19 | З вихідної послідовності цілих чисел побудувати бінарне дерево пошуку. Вивести покажчик P на останню вершину дерева з максимальним непарним значенням (вершини перебирати в інфіксному порядку). Якщо дерево не має вершин з непарними значеннями, то вивести null |
| 20 | З вихідної послідовності дійсних чисел побудувати бінарне дерево пошуку. Використовуючи будь– який із засобів обхода дерева, вивести значення усіх вершин рівня L, а також іх кількість N (якщо дерево не має вершин рівня L, то вивести 0) |
| 21 | З вихідної послідовності дійсних чисел побудувати бінарне дерево пошуку. Вивести кількість вузлів дерева, значення яких дорівнює заданому числу K |
| 22 | З вихідної послідовності цілих чисел побудувати бінарне дерево пошуку. Вивести на екран середньоарифметичне значення усіх нащадків вузлів рівня L |
| 23 | З вихідної послідовності дійсних чисел побудувати бінарне дерево пошуку. Для кожного рівня даного дерева, починаючи з нульового, вивести суму значень вершин, які знаходяться на цьому рівні |
| 24 | З вихідної послідовності дійсних чисел побудувати бінарне дерево пошуку. Вивести значення усіх вершин дерева у постфіксному порядку (спочатку вивести значення лівого піддерева у постфіксному порядку, потім – значення правого піддерева у постфіксному порядку, потім – значення кореня) |
| 25 | З вихідної послідовності цілих чисел побудувати бінарне дерево пошуку. Вивести значення усіх вершин дерева у префіксному порядку (спочатку вивести значення кореня, потім — значення лівого піддерева у префіксному порядку, потім — значення правого піддерева у префіксному порядку) |
| 26 | З вихідної послідовності цілих чисел побудувати бінарне дерево пошуку. Визначити чи є у цьому дереві хоча б одна пара вершин з однаковими значеннями |
| 27 | З вихідної послідовності цілих чисел побудувати бінарне дерево пошуку. Знайти різницю у кількісті рівней правого та лівого нащадків кореня |
| 28 | З вихідної послідовності цілих чисел побудувати бінарне дерево пошуку. Додати новий елемент до бінарного дерева, вызначити його рівень та номер |

| № вар. | Завдання |
|---|---|
| 29 | З вихідної послідовності дійсних чисел побудувати бінарне дерево пошуку. Видалити заданий елемент з бінарного дерева, визначити кількість рівнів нового дерева |
| 30 | З вихідної послідовності дійсних чисел побудувати бінарне дерево пошуку. Вивести всі значання вузлів заданого рівня L |

# Лабораторна робота № 5

# АЛГОРИТМИ ПОШУКУ

**Мета:** ознайомитися з простими і складними алгоритмами пошуку заданого елемента у масиві та підрядка у рядку.

## 5.1 Пошук заданого елемента у масиві

### *Алгоритм послідовного пошуку*

```
//повертає позицію елемента, якщо він є в масиві.
// повертає -1, якщо його немає
int find_el(double* arr, int count, double element)
{ for (int i=0; i<count; i++)
      {if (arr[i]==element) {// знайшли.
              return i;
        }
}
return -1;
}
```

Асимптотика роботи алгоритма O(n) – у циклі n операцій порівняння.

### *Алгоритм бінарного пошуку*

Алгоритм бінарного пошуку застосовується у випадку, коли необхідно здійснити пошук на великих масивах багаторазово. Асимптотика роботи самого алгоритма $O(\log_2 n)$. Результати показані на рис. 5.1.

```
#include<ctime>
using namespace std;
// функція алгоритма бінарного пошуку
int Search_Binary(int arr[], int left, int right, int key) {
     int midd = 0;
     while (1)  {
          midd = (left + right) / 2;

          if (key < arr[midd])// якщо шукане менше значення у комірці
               right = midd - 1;// зміщення правої межі пошуку
          else   // якщо шукане більше значення у комірці
               if (key > arr[midd])
               left = midd + 1;   // зміщнння левої межі пошуку
          else                       // інакше (значення дорівнює)
               return midd;           // функція повертає індекс комірки
```

```
          if (left > right)          //якщо межі зімкнулися
               return -1;
     }
}
//упорядкування масиву для бінарного пошуку
void sort_b(int arr[], int n) {
int i, j, imin, tmp;
   for (i = 0; i < n - 1;i++) {
        imin = i;
        for (j = i + 1; j < n; j++)
          if (arr[imin]>arr[j]) imin = j;
        tmp = arr[i];
        arr[i] = arr[imin];
        arr[imin] = tmp;
   }
}

int main()
{
     const int SIZE = 50;
     int array[SIZE] = {};
     int key = 0;
     int index = 0; // індекс комірки з шуканим значенням
      cout<<"Сформувати масив"<<endl;
     for ( i = 0; i < SIZE; i+=10){
          for ( j = i; j < i+10;j++)
               { array[i] =rand()%100;
                 cout << array[j] << "  ";}
          cout << endl;
     }
     cout << endl;
     cout << "Відсортувати масив"<<endl;
     sort_b(array, SIZE);
     for ( i = 0; i < SIZE; i+=10)   {
          for ( j = i; j < i+10;j++)
               cout << array[j] << "  ";
          cout << endl;
     }
     cout << "\n\n Введіть деяке число: ";
     cin >> key;
     index = Search_Binary(array, 0, SIZE, key);
     if (index >= 0)
          cout << "Вказане число знаходиться у комірці з індексом: " <<
          index << "\n\n";
     else
          cout << "У масиві немає такого числа!\n\n";
     return 0;
}
```

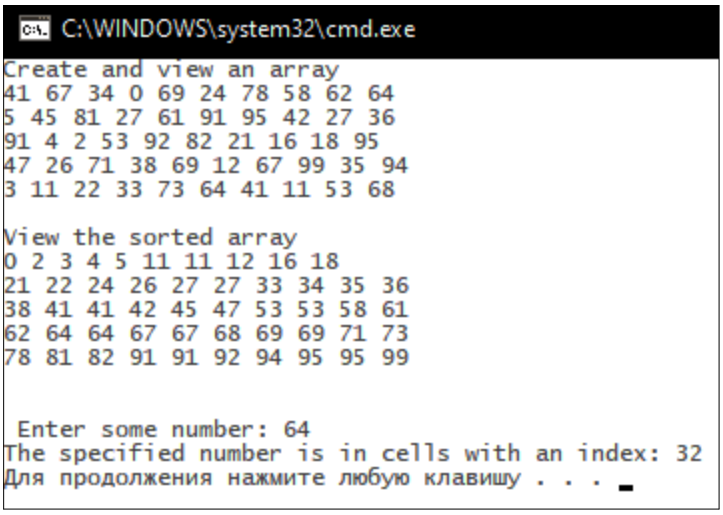



Рисунок 5.1 –

## 5.2 Пошук підрядка у рядку

### *Прямий пошук підрядка у рядку*

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

//функції прямого пошуку підрядка у рядку
int DirectSearch(char *string, char *substring){
	int sl, ssl;
	int res = -1;
	sl = strlen(string);
	ssl = strlen(substring);
	if (sl == 0)
		cout << "Невірно заданий рядок\n";
	else if (ssl == 0)
		cout << "Невірно заданий підрядок\n";
	else
		for (int i = 0; i < sl - ssl + 1; i++)
			for (int j = 0; j < ssl; j++)
				if (substring[j] != string[i + j])
					break;
				else if (j == ssl - 1){
					res = i;
					break;
				}
	return res;
}
```

Для алгоритма прямого пошуку необхідний час порядку O(n-m)*m, де n та m – довжина рядка та підрядка відповідно

### *Алгоритм Кнута-Моріса-Прата*

```
int KMPSearch(char *string, char *substring){
	int  sl, ssl;
```

```
      int res = -1;
      sl = strlen(string);
      ssl = strlen(substring);
      if (sl == 0)
            cout << "Невірно заданий рядок\n";
      else if (ssl == 0)
            cout << "Невірно заданий підрядок\n";
      else {
            int  i, j = 0, k = -1;
            int  *d;
            d = new int[1000];//масив довжини префіксів
            d[0] = -1;
// заповнюємо масив довжини префіксів для зразка
            while (j < ssl - 1) {
                  while (k >= 0 && substring[j] != substring[k])
                        k = d[k];
                  j++;
                  k++;
                  if (substring[j] == substring[k])
                        d[j] = d[k];
                  else
                        d[j] = k;
            }

            i = 0;     j = 0;
// пошук входження підрядка
            while (j < ssl && i < sl){
                  while (j >= 0 && string[i] != substring[j])
                        j = d[j];
                  i++;
                  j++;
            }
            delete[] d;
//індекс першого входження підрядка
            res = j == ssl ? i - ssl : -1;
      }
      return res;
}
```

Приклад застосування алгоритмів «Прямого пошуку» та «Алгоритму КМП» для пошуку індексу входження підрядка *sb* у заданий рядок *st*. Результат показаний на рис. 5.2.

```
int main()
{
      char *st = "A black hole is a region of spacetime exhibiting such
strong gravitational effects that nothing—not even particles and
electromagnetic radiation such as light—can escape from inside it.The
theory of general relativity predicts that a sufficiently compact mass
can deform spacetime to form a black hole.";
      char *sb = "it";
```

```
    int ind1 = DirectSearch(st, sb);
    cout << "Result DirectSearch =" << ind1 << endl;

    int ind2 = KMPSearch(st, sb);

    cout << "Result KMPSearch =" << ind2 << endl;
    return 0;
}
```

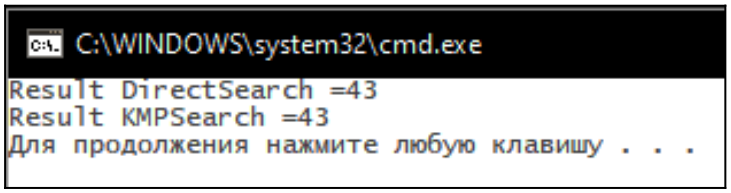



Рисунок 5.2 – Результати роботи програми прикладу 5.2

Для алгоритму КМП час роботи оцінюється як O(m+n), де n та m – довжина рядка та підрядка відповідно, що значно ліпше, ніж для алгоритму прямого пошуку.

## Питання для самоконтролю

1. Чим можна пояснити різноманіття алгоритмів пошуку в лінійних структурах?
2. У чому переваги пошуку з бар'єром порівнянно з послідовним пошуком?
3. Знаходження якого за порядком елемента в лінійній множині (першого, останнього) гарантує алгоритм послідовного пошуку? Як в цьому випадку повинен бути виконаний перегляд?
4. Знаходження якого за порядком елемента в лінійній множині (першого, останнього) гарантує алгоритм бінарного пошуку? Відповідь обґрунтуйте.
5. Коли алгоритм КМП буде ефективний?

## Завдання до лабораторної роботи

**1.** Відповісти на питання для самоконтролю.

**2.** Написати програму, яка реалізує алгоритми пошуку заданого елемента для рішення завдання свого варіанта у табл. 5.1.

**3.** Для заповнення масиву треба використовувати функцію rand() – генератор випадкових чисел. Передбачити вивід повідомлення у випадку відсутності елемента у масиві. Текстовий рядок зчитується з файла.

**4.** Написати програму за заданням свого варіанта, яка продемонструє роботу алгоритмів бібліотек STL – find(), find_if(), find_first(), find_end().

Таблиця 5.1

| Варіанти | **Завдання** |
|---|---|
| 1 - 5 | **1.** Створити масив дійсних чисел з 1000 елементів в межах від 0 до 5. Знайти за допомогою алгоритму бінарного пошуку заданий елемент.<br>**2.** Знайти заданий підрядок у рядку розміром не менше 300 символів за допомогою алгоритму прямого пошуку |
| 6 - 10 | 1. Створити масив цілих чисел з 5000 елементів в межах від 0 до 500. Знайти за допомогою алгоритму послідовного пошуку заданий елемент.<br>2. Знайти заданий підрядок у рядку розміром не менше 300 символів за допомогою алгоритму КМП. Рядок зчитується з файла<br>3. |
| Варіанти | **Завдання** |
| 11 - 15 | 1. Створити масив дійсних чисел з 1000 елементів в межах від -100 до +100. Знайти за допомогою алгоритму бінарного пошуку заданий елемент.<br>2. Знайти заданий підрядок у рядку розміром не менше 300 символів за допомогою алгоритму прямого пошуку |
| 16 - 20 | 1. Створити масив цілих чисел з 5000 елементів в межах від -200 до 1000. Знайти за допомогою алгоритму послідовного пошуку заданий елемент.<br>2. Знайти заданий підрядок у рядку розміром не менше 300 символів за допомогою алгоритму КМП |
| 21 - 25 | Створити масив дійсних чисел з 3000 елементів в межах від -10 до +500. Знайти за допомогою алгоритму бінарного пошуку заданий елемент.<br>Знайти заданий підрядок у рядку розміром не менше 300 символів за допомогою алгоритму КМП |
| 26 - 30 | 3. Створити масив цілих чисел з 7000 елементів в межах від -300 до 800. Знайти за допомогою алгоритму послідовного пошуку заданий елемент.<br>Знайти заданий підрядок у рядку розміром не менше 300 символів за допомогою алгоритму прямого пошуку |

# Лабораторна робота № 6

# АЛГОРИТМИ ПРОСТОГО ТА ХОРОШЕГО СОРТУВАННЯ

**Мета:** набути практичних навичок застосування простих та добрих алгоритмів сортування.

## 6.1 Алгоритми простого сортування

Асимптотика часу роботи алгоритмів простого сортування $O(n^2)$.

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

// алгоритм вибору

void select_sort(int *a,int n){
     int i, j, min_ind;
     for (i = 0; i < n - 1;i++)
     {
         min_ind =i;
          for (j = i + 1; j < n; j++)
               if (a[j]<a[min_ind]) min_ind = j;
          swap(a[i], a[min_ind]);
     }
}

void swap(int a,int b){
     int tmp = a;
     a = b;
     b = tmp;
}
```

**//алгоритм вставки**

```
void insert_sort(int *a,int n){
     int i, j,tmp;
     for (i = 0; i < n;i++)
     {
          tmp = a[i];
          for (j = i; j > 0 && tmp < a[j - 1]; j--)
               a[j] = a[j - 1];
          a[j] = tmp;
     }
}
```

**//алгоритм бульбашкового сортування**

```
void bubble_sort(int *a,int n)
{
     int i, j = 0;
     for (i = 0; i < n - 1; i++)
          for (j = i + 1; j < n; j++)
               if (a[i]>a[j])
                    swap(a[i], a[j]);
}

int main()
{
     const int SIZE = 2000;
     int array[SIZE] = {};

     for (int i = 0; i < SIZE; i++) // заповнюємо та переглядаємо масив
     {
          array[i] = rand()%100+1;
          cout << array[i] << " | ";
     }
     cout << endl;
     unsigned int start_time = clock();
     select_sort(array, SIZE); //виклик алгоритму сортування вибором
     unsigned int end_time = clock(); // кінцевий час
     unsigned int search_time = end_time - start_time; // шуканий час

     for (int i = 0; i < SIZE; i++)
          cout << array[i] << " | ";

     cout <<"time 1="<<search_time<< endl;
     return 0;
}
```

***Оцінка часу роботи алгоритму***

Найпростіший варіант оцінки часу роботи алгоритму – це обчислити різницю між початковим та кінцевим часом його роботи. Наприклад:

```
#include <ctime>
//          ...
    unsigned int start_time =  clock(); // початковий час
   …… //  фрагмент коду алгоритму
    unsigned int end_time = clock(); // кінцевий час
    unsigned int search_time = end_time - start_time; // різниця часу
```

Для того, щоб знайти час роботи програми, потрібно скористатися функцію `clock()`. Прототип функції `clock()` знаходиться у заголовному файлі <ctime>, який потрібно підключити. Функція `clock()` повертає значення часу у мілісекундах (1 с = 1000 млс). Причому відлік часу починається з моменту запуску програми. Якщо треба заміряти час роботи усієї програми, то в кінці програми, перед оператором `return 0`; треба запустити функцію `clock()`.

### 6.2 Алгоритми хорошего сортування

Асимптотика часу роботи алгоритмів доброго сортування O(n log n).

#### *6.2.1 Пірамідальне сортування*

***Двійкова купа ( піраміда*** або ***сортувальне дерево)***– таке двійкове дерево, для якого виконується три умови:

1. значення у будь–якої вершині не менше, ніж значення її нащадків.;
2. глибина усього листя (відстань до кореня) відрізняється не більше, чим на 1прошарок;
3. останній прошарок заповнюється зліва направо без «дірок».

Існують також купи, де значення у будь–якій вершині, навпаки, не більше, ніж значення її нащадків. Такі купи називаються min-heap, а купи, які описані вище — max-heap. Далі будемо розглядати лише max-heap. Усі дії з min-heap виконуються аналогічно.

Основні операції для реалізації алгоритму сортування купою:

1. Процедура *Heapify* дозволяє підтримувати основні властивості купи (час роботи O(log n)).
2. Процедура *Build-Heap* будує купу з не відсортованого масиву за час *O*( n).
3. Процедура ***Heapsort*** упорядковує масив, незастосовуючи додаткової пам’яті за час *O*(log n).

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

int n,heap_lenght;
int Left(int i)
```

```
{ return 2 * i+1; }
int Right(int i)
{ return 2 * i + 2; }

//функція підтримує основну властивість купи
void Heapify(int *a, int i)
{ int max, l = Left(i);
int r = Right(i);
if (l<heap_lenght&&a[l]>a[i]) max = l;
else max = i;
if (r<heap_lenght&&a[r]>a[max]) max = r;
if (max != i)
{
swap(a[i], a[max]);
Heapify(a, max);
   }
}
//функція побудови купи
void Build_Heap(int *a)
{
heap_lenght = n;
for (int i = n / 2-1; i >= 0; i--)
Heapify(a, i);
}
//функція, яка реалізує алгоритм пірамідального сортування
void Heap_Sort(int *a)
{  Build_Heap(a);
for (int i = n-1; i > 0; i--)
{  swap(a[0], a[i]);
heap_lenght = heap_lenght - 1;
Heapify(a, 0);
  }
}

int main()
{
int a[9] = { 3, 5, 7, 2, 11, 9, 4, 12, 1 };
heap_lenght=n = 9;
Heap_Sort(a);
for (int i = 0; i < n; i++)
cout << a[i] << "\t";
cout << endl;
return 0;
}
```

### 6.2.2 Сортування Хоара (QUICK SORT)

Швидке сортування – це загальна назва низки алгоритмів, які відображають різні підходи до отримання критичного параметру, який впливає на ефективність алгоритма. Цей параметр називається ***опорним елементом***.

***Алгоритм швидкого сортування:***

1. Виберемо опорний елемент масиву.
2. Поділимо масив на дві частини відносно опорного, так щоб елементи у лівій частині мали значення менше, ніж у правій.
3. Застосуємо цю процедуру до лівої та правої частин рекурсивно.

```
//функція, яка реалізує алгоритм швидкого сортування, from, to – межі
сортування
void QuickSort(int A[], int from, int to) {
     int x, i, j, temp;
     if (from >= to) return;
     i = from;
     j = to;
     x = A[from ]; //опорний елемент
     while (i <= j)  {
          while (A[i] < x) i++;
           while (A[j] > x) j--;
           if (i <= j) {
                      temp = A[i]; A[i] = A[j]; A[j] = temp;
                      i++;
                      j--;
                          }
     }
QuickSort(A, from, j);
QuickSort(A, i, to);
}
```

### *6.2.3 Алгоритм сортування злиттям*

Ідея алгоритму сортування злиттям полягає у наступному: масив поділяється на дві половинки меншого розміру, потім ми упорядковуємо ці дві половинки. Після цього ці половинки з'єднуються. Рекурсивне розбиття задачі на частини відбувається доки розмір масиву не стане дорівнювати 1, який, очевидно, є відсортованим. Нетривіальним є з'єднання відсортованих масивів до одного. З'єднання виконується за допомогою допоміжної процедури **Merge(A,p,q,r)**. Параметрами цієї процедури є масив A та числа p,q,r , які вказують межі зливаних частин. Процедура передбачає, що p<=q<r і частини A[p..q] та A[q+1..r] вже відсортовані, ці частини зливаються (merges) у одну частину A[p..r]. Процедуру легко пояснити на ігральних картах. Треба взяти дві вже відсортовані пачки карт та зробити з них одну відсортовану, для цього беремо з кожної пачки найменшу карту і додаємо її «сорочкою» донизу.

```
//функція сортування злиттям
void merge_sort(int a[],int p, int r)
{
if (p == r)
return;
if (r - p == 1)
      { if (a[r] < a[p]) swap(a[r], a[p]); return; }
```

```
int m = (r + p) / 2;
merge_sort(a, p, m);
merge_sort(a, m + 1, r);
 int xl = p, xr = m + 1, cur = 0;
int buff[100];
while (r - p + 1 != cur)
{ if (xl > m)   buff[cur++] = a[xr++];
      else if (xr > r)buff[cur++] = a[xl++];
            else if (a[xl] > a[xr]) buff[cur++] = a[xr++];
                  else buff[cur++] = a[xl++]; }
for (int i = 0; i < cur; i++)
a[i + p] = buff[i];
}
```

## Питання для самоконтролю

1. Чим можна пояснити різноманіття алгоритмів сортування?
2. Чому на даний момент не існує універсального алгоритму сортування?
3. За рахунок чого в алгоритмах швидкого сортування відбувається виграш при виконанні операцій порівняння і перестановок?
4. Які з перерахованих алгоритмів найбільш ефективні на майже відсортованих масивах: бінарне пірамідальне сортування, сортування злиттям, сортування Шелла і сортування Хоара? За рахунок чого відбувається виграш?
5. Чому алгоритми швидкого сортування не дають великого виграшу при малих розмірах масивів?
6. Як визначити, яким алгоритмом сортування віддати перевагу при розв'язанні задачі?

## Завдання до лабораторної роботи

1. Відповісти на питання для самоконтролю.
2. Створити масив згідно зі своїм варіантом табл. 6.1, відсортувати масив за допомогою трьох швидких алгоритмів сортування, визначити час роботи цих алгоритмів.
3. Згідно зі своїм варіантом визначити для 10 масивів заданого розміру (n = 40, 70, 100, 1000, 10000, 500000, $10^6$, 5 $10^7$, 5 $10^8$, $10^9$) час роботи T(n).
   Побудувати графік відносної швидкості роботи алгоритму як функції розміру масиву за формулою:
   $k(n)=10^9\frac{T(n)}{n\cdot\log n}$
   Зробити висновки про швидкість роботи заданого алгоритму від розміру масиву.

Таблиця 6.1

| **Варіанти** | **Завдання** |
|---|---|
| **1 - 5** | Створити масив дійсних чисел з n елементів у межах від 0 до 500 . Відсортувати його за допомогою алгоритму пірамідального |

| | |
|---|---|
| | сортування |
| **6 - 10** | Створити масив цілих чисел з n елементів у межах від 0 до 50000. Відсортувати його за допомогою алгоритму швидкого сортування |
| **11 - 15** | Створити масив дійсних чисел з n елементів у межах від -100 до +100. Відсортувати його за допомогою алгоритму злиттям |
| **16 - 20** | Створити масив цілих чисел з n елементів у межах від -200 до 10000. Відсортувати його за допомогою алгоритму швидкого сортування |
| **21 - 25** | Створити масив дійсних чисел з n елементів у межах від -10 до +1000. Відсортувати його за допомогою алгоритму пірамідального сортування |
| **26 - 30** | Створити масив цілих чисел з n елементів у межах від -10000 до 10000. Відсортувати його за допомогою алгоритма злиттям. |

# Лабораторна робота № 7

# ПРОГРАМУВАННЯ ЗАДАЧ ОПТИМІЗАЦІЇ З ГРАФАМИ: АЛГОРИТМ ФЛОЙДА

**Мета:** набути практичних навичок у пошуку найкоротших шляхів за допомогою алгоритмів на графах.

### *Алгоритм Флойда*

1. Пронумерувати вершини графа від 1 до $N$ цілими числами, визначити матрицю $D^0$, кожен елемент $d_{i,j}$ якої є довжина найкоротшої дуги між вершинами $i$ та $j$. Якщо такої дуги немає, покласти значення елемента рівним ∞. Крім того, покласти значення діагонального елемента $d_{i,i}$ рівним 0.
2. Для цілого $m$, яке послідовно набуває значення 1 ... $N$, визначити за елементами матриці $D^{m-1}$ елементи $D^m$.
3. Алгоритм закінчується отриманням матриці всіх найкоротших шляхів $D^N$, $N$ − число вершин графа.

Щоб записати найкоротший шлях у вигляді маршруту з переліком вузлів, треба розрахувати матрицю $C_{i,j}$ передостаннього вузла на найкоротших шляхах між вузлами $i$ та $j$, тобто на перетині $i$ та $j$ вузлів буде записаний номер передостаннього вузла на маршруті від $i$ до $j$. Для цього треба матрицю розміром $N \times N$, елементи рядків якої на початку мають значення номера рядка, а потім у циклах пошуку найкоротшого шляху між вузлами $i$ та $j$ елементи цієї матриці отримають значення передостаннього вузла на шляху.

```
//функція алгоритму Флойда
     void floid(int **A, int **C, int n)
  {
int i, j, k;
for (k = 0; k < n; k++)
     for (i = 0; i < n; i++)
          for (j = 0; j<n; j++)
               if (A[i][j]>A[i][k] + A[k][j])
               {
                    A[i][j] = A[i][k] + A[k][j];
                    C[i][j] = C[k][j];
               }
   }
```

Приклад: визначення найкоротшого шляху за допомогою алгоритму Флойда.

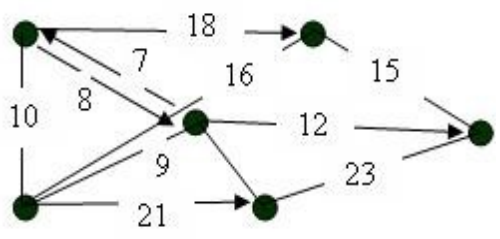
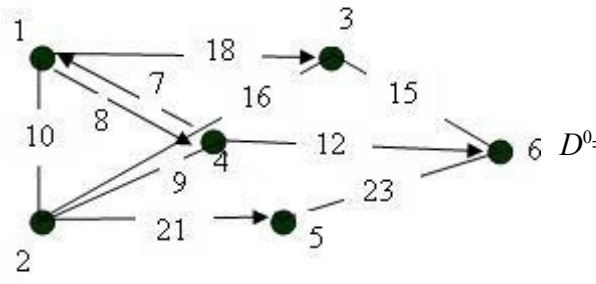



Рисунок 7.1− Орієнтований граф

Необхідно знайти найкоротший шлях між кожною парою вершин у графі, який показано на рис. 7.1.

Пронумерувати усі вершини графа та

скласти матрицю довжини найкоротших дуг $D^0$, у випадку, якщо дуги поміж вершиною $i$ та $j$ не існує, елементу $d_{i,j}$ матриці надається значення ∞. Вихідний граф з пронумерованими вершинами показано на рис. 7.2.




$$D^0=\begin{pmatrix} 0 & 10 & 18 & 8 & \infty & \infty \\ 10 & 0 & 16 & 9 & 21 & \infty \\ \infty & 16 & 0 & \infty & \infty & 15 \\ 7 & 9 & \infty & 0 & \infty & 12 \\ \infty & \infty & \infty & \infty & 0 & 23 \\ \infty & \infty & 15 & \infty & 23 & 0 \end{pmatrix}$$

Рисунок 7.2– Вихідний граф з пронумерованими вершинами

На підставі матриці $D^0$ розраховувати послідовно усі елементи матриці $D^1$. Для цього ми використаємо рекурентне співвідношення

$$d_{i,j}^{1}=\min\{d_{i,1}^{0}+ d_{1,j}^{0};d_{ij}^{0}\}.$$

$d_{1,1}^{1}= \min\{d_{1,1}^{0}+d_{1,1}^{0}d_{1,1}^{0}\}= \min\{0+0; 0\}=0$
$d_{1,2}^{1}= \min\{d_{1,1}^{0}+d_{1,2}^{0}d_{1,2}^{0}\}= \min\{0+10; 10\}=10$
$d_{1,3}^{1}= \min\{d_{1,1}^{0}+d_{1,3}^{0}d_{1,3}^{0}\}= \min\{0+18; 18\}=18$
$d_{1,4}^{1}= \min\{d_{1,1}^{0}+ d_{1,4}^{0}d_{1,4}^{0}\}=\min\{0+8; 8\}=8$
$d_{1,5}^{1}= \min\{d_{1,1}^{0}+d_{1,5}^{0}d_{1,5}^{0}\}= \min\{0+\infty; \infty\}=\infty$
$d_{1,6}^{1}= \min\{d_{1,1}^{0}+d_{1,6}^{0}d_{1,6}^{0}\}= \min\{0+\infty; \infty\}=\infty$
$d_{2,1}^{1}= \min\{d_{2,1}^{0}+d_{1,1}^{0}d_{2,1}^{0}\}= \min\{10+0; 10\}=10$
$d_{2,2}^{1}= \min\{d_{2,1}^{0}+d_{1,2}^{0}d_{2,2}^{0}\}= \min\{10+10; 0\}=0$
$d_{2,3}^{1}= \min\{d_{2,1}^{0}+d_{1,3}^{0}d_{2,3}^{0}\}= \min\{10+18; 16\}=16$
$d_{2,4}^{1}= \min\{d_{2,1}^{0}+d_{1,4}^{0}d_{2,4}^{0}\}= \min\{10+8; 9\}=9$
$d_{2,5}^{1}= \min\{d_{2,1}^{0}+d_{1,5}^{0}d_{2,5}^{0}\}= \min\{10+\infty; 21\}=21$
$d_{2,6}^{1}= \min\{d_{2,1}^{0}+d_{1,6}^{0}d_{2,6}^{0}\}= \min\{10+\infty; \infty\}=\infty$
$d_{3,1}^{1}= \min\{d_{3,1}^{0}+d_{1,1}^{0}d_{3,1}^{0}\}= \min\{\infty+0; \infty\}=\infty$
$d_{3,2}^{1}= \min\{d_{3,1}^{0}+d_{1,2}^{0}d_{3,2}^{0}\}= \min\{\infty+10; 16\}=16$

$d_{3,3}^1 = \min\{d_{3,1}^0 + d_{1,3}^0 d_{3,3}^0\} = \min\{\infty+18;\ 0\} = 0$
$d_{3,4}^1 = \min\{d_{3,1}^0 + d_{1,4}^0 d_{3,4}^0\} = \min\{\infty+8;\ \infty\} = \infty$
$d_{3,5}^1 = \min\{d_{3,1}^0 + d_{1,5}^0 d_{3,5}^0\} = \min\{\infty+\infty;\ \infty\} = \infty$
$d_{3,6}^1 = \min\{d_{3,1}^0 + d_{1,6}^0 d_{3,6}^0\} = \min\{\infty+\infty;\ 15\} = 15$
$d_{4,1}^1 = \min\{d_{4,1}^0 + d_{1,1}^0 d_{4,1}^0\} = \min\{7+0;\ 7\} = 7$
$d_{4,2}^1 = \min\{d_{4,1}^0 + d_{1,2}^0 d_{4,2}^0\} = \min\{7+10;\ 9\} = 9$
$d_{4,3}^1 = \min\{d_{4,1}^0 + d_{1,3}^0 d_{4,3}^0\} = \min\{7+18;\ \infty\} = 25$
$d_{4,4}^1 = \min\{d_{4,1}^0 + d_{1,4}^0 d_{4,4}^0\} = \min\{7+8;\ 0\} = 0$
$d_{4,5}^1 = \min\{d_{4,1}^0 + d_{1,5}^0 d_{4,5}^0\} = \min\{7+\infty;\ \infty\} = \infty$
$d_{4,6}^1 = \min\{d_{4,1}^0 + d_{1,6}^0 d_{4,6}^0\} = \min\{7+\infty;\ 12\} = 12$
$d_{5,1}^1 = \min\{d_{5,1}^0 + d_{1,1}^0 d_{5,1}^0\} = \min\{\infty+0;\ \infty\} = \infty$
$d_{5,2}^1 = \min\{d_{5,1}^0 + d_{1,2}^0 d_{5,2}^0\} = \min\{\infty+10;\ \infty\} = \infty$
$d_{5,3}^1 = \min\{d_{5,1}^0 + d_{1,3}^0 d_{5,3}^0\} = \min\{\infty+18;\ \infty\} = \infty$
$d_{5,4}^1 = \min\{d_{5,1}^0 + d_{1,4}^0 d_{5,4}^0\} = \min\{\infty+8;\ \infty\} = \infty$
$d_{5,5}^1 = \min\{d_{5,1}^0 + d_{1,5}^0 d_{5,5}^0\} = \min\{\infty+\infty;\ 0\} = 0$
$d_{5,6}^1 = \min\{d_{5,1}^0 + d_{1,6}^0 d_{5,6}^0\} = \min\{\infty+\infty;\ 23\} = 23$
$d_{6,1}^1 = \min\{d_{6,1}^0 + d_{1,1}^0 d_{6,1}^0\} = \min\{\infty+0;\ \infty\} = \infty$
$d_{6,2}^1 = \min\{d_{6,1}^0 + d_{1,2}^0 d_{6,2}^0\} = \min\{\infty+10;\ \infty\} = \infty$
$d_{6,3}^1 = \min\{d_{6,1}^0 + d_{1,3}^0 d_{6,3}^0\} = \min\{\infty+18;\ 15\} = 15$
$d_{6,4}^1 = \min\{d_{6,1}^0 + d_{1,4}^0 d_{6,4}^0\} = \min\{\infty+8;\ \infty\} = \infty$
$d_{6,5}^1 = \min\{d_{6,1}^0 + d_{1,5}^0 d_{6,5}^0\} = \min\{\infty+\infty;\ 23\} = 23$
$d_{6,6}^1 = \min\{d_{6,1}^0 + d_{1,6}^0 d_{6,6}^0\} = \min\{\infty+\infty;\ 0\} = 0$

Записати матрицю $D^1$, задаючи до неї розраховані елементи.

$$D^1 = \begin{matrix} 0 & 10 & 18 & 8 & \infty & \infty \\ 10 & 0 & 16 & 9 & 21 & \infty \\ \infty & 16 & 0 & \infty & \infty & 15 \\ 7 & 9 & 25 & 0 & \infty & 12 \\ \infty & \infty & \infty & \infty & 0 & 23 \\ \infty & \infty & 15 & \infty & 23 & 0 \end{matrix}$$

На основі матриці $D^1$ аналогічно обчислити послідовно усі елементи матриці $D^2$. Для цього використати рекурентне співвідношення
$d_{i,j}^2 = \min\{\ d_{i,2}^1 + d_{2,j}^1;\ d_{i,j}^1\}$.

$$D^2 = \begin{matrix} 0 & 10 & 18 & 8 \\ 10 & 0 & 16 & 9 \\ 26 & 16 & 0 & 25 \\ 7 & 9 & 25 & 0 \\ \infty & \infty & \infty & \infty \\ \infty & \infty & 15 & \infty \end{matrix} \quad D^3 = \begin{matrix} 0 & 10 & 18 & 8 & 31 & 33 \\ 10 & 0 & 16 & 9 & 21 & 31 \\ 26 & 16 & 0 & 25 & 37 & 15 \\ 7 & 9 & 25 & 0 & 30 & 12 \\ \infty & \infty & \infty & \infty & 0 & 23 \\ 41 & 31 & 15 & 40 & 23 & 0 \end{matrix}$$

$$D^4 = \begin{matrix} 0 & 10 & 18 & 8 & 31 & 20 \\ 10 & 0 & 16 & 9 & 21 & 21 \\ 26 & 16 & 0 & 25 & 37 & 15 \\ 7 & 9 & 25 & 0 & 30 & 12 \\ \infty & \infty & \infty & \infty & 0 & 23 \\ 41 & 31 & 15 & 40 & 23 & 0 \end{matrix}$$

$$D^6 = \begin{matrix} 0 & 10 & 18 & 8 & 31 & 20 \\ 10 & 0 & 16 & 9 & 21 & 21 \\ 26 & 16 & 0 & 25 & 37 & 15 \\ 7 & 9 & 25 & 0 & 30 & 12 \\ 64 & 54 & 38 & 63 & 0 & 23 \\ 41 & 31 & 15 & 40 & 23 & 0 \end{matrix}$$

$$D^5 = \begin{matrix} 0 & 10 & 18 & 8 & 31 & 20 \\ 10 & 0 & 16 & 9 & 21 & 21 \\ 26 & 16 & 0 & 25 & 37 & 15 \\ 7 & 9 & 25 & 0 & 30 & 12 \\ \infty & \infty & \infty & \infty & 0 & 23 \\ 41 & 31 & 15 & 40 & 23 & 0 \end{matrix}$$

На основі матриці $D^2$ обчислити послідовно усі елементи матриці $D^3$, а далі матриці $D^4$, $D^5$, $D^6$. Отже, ми отримали матрицю довжин найкоротших шляхів між кожною парою вершин графа. Нижче надано таблицю шляхів. Кожний елемент $d_{i\text{-}j}$ таблиці – це шлях з вершини $i$ до вершини $j$:

| **Таблиця шляхів** ***i\j*** | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | - | $d_{1\text{-}2}$=1-2 | $d_{1\text{-}3}$=1-3 | $d_{1\text{-}4}$=1-4 | $d_{1\text{-}5}$=1-2-5 | $d_{1\text{-}6}$=1-4-6 |
| 2 | $d_{2\text{-}1}$=2-1 | - | $d_{2\text{-}3}$=2-3 | $d_{2\text{-}4}$=2-4 | $d_{2\text{-}5}$=2-5 | $d_{2\text{-}6}$=2-4-6 |
| 3 | $d_{3\text{-}1}$=3-2-1 | $d_{3\text{-}2}$=3-2 | - | $d_{3\text{-}4}$=3-2-4 | $d_{3\text{-}5}$=3-2-5 | $d_{3\text{-}6}$=3-6 |
| 4 | $d_{4\text{-}1}$=4-1 | $d_{4\text{-}2}$=4-2 | $d_{4\text{-}3}$=4-1-3 | - | $d_{4\text{-}5}$=4-2-5 | $d_{4\text{-}6}$=4-6 |
| 5 | $d_{5\text{-}1}$=5-6-3-2-1 | $d_{5\text{-}2}$=5-6-3-2 | $d_{5\text{-}3}$=5-6-3 | $d_{5\text{-}4}$=5-6-3-2-4 | - | $d_{5\text{-}6}$=5-6 |
| 6 | $d_{6\text{-}1}$=6-3-2-1 | $d_{6\text{-}2}$=6-3-2 | $d_{6\text{-}3}$=6-3 | $d_{6\text{-}4}$=6-3-2-4 | $d_{6\text{-}5}$=6-5 | - |

## Питання для самоконтролю

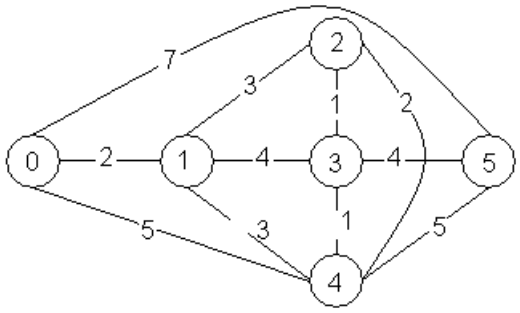
1. З якими видами графів працюють алгоритми Дейкстри і Флойда?
2. Як від надання графа залежить ефективність алгоритму його обходу?
3. За рахунок чого пошук у ширину є досить ресурсомістким алгоритмом?
4. У чому переваги алгоритмів обходу графа в ширину?

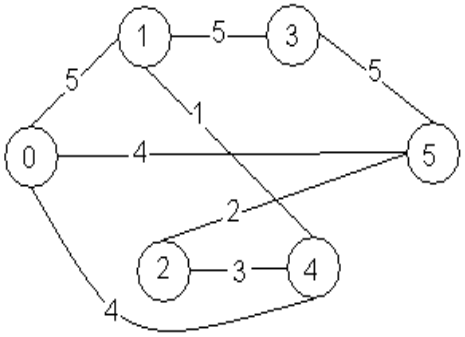
## Завдання до лабораторної роботи

1. Відповісти на питання для самоконтролю
2. Згідно свого варіанта (таблиця 7.1) розрахувати матрицю довжин найкоротших шляхів та створити таблицю шляхів згідно з прикладом, який розглянут вище.
3. Написати програму обчислення матриці довжин найкоротших шляхів за допомогою алгоритма Флойда. Порівняти результати цього обчислення.

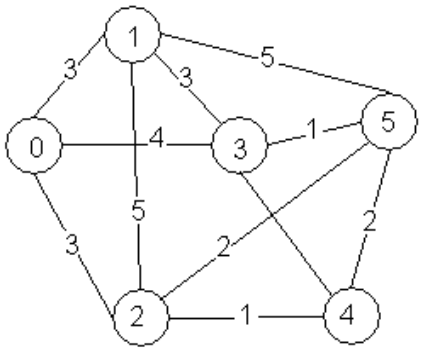
Таблица 7.1

1

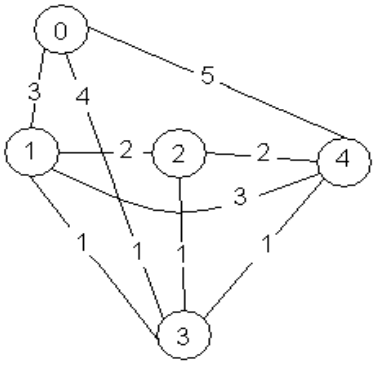

0
1
2
3
4
5
7
3
1
2
2
4
4
5
3
1
5


1
3
0
5
2
4
5
5
1
4
2
3
4

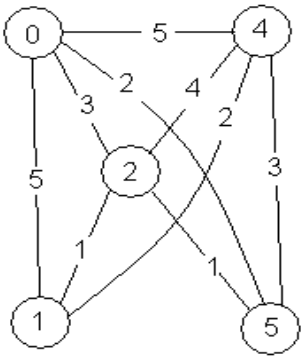
3


1
0
3
5
2
4
3
5
3
4
1
5
3
2
2
1

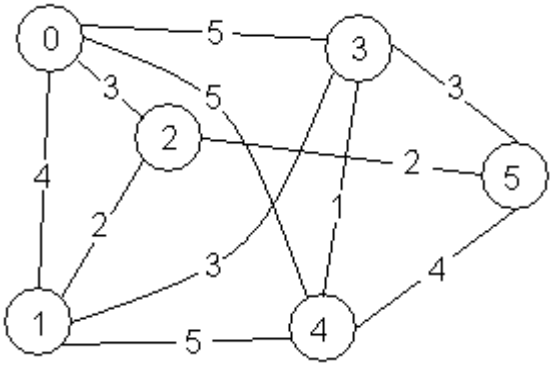
4


0
1
2
4
3
5
3
4
2
2
3
1
1
1
1

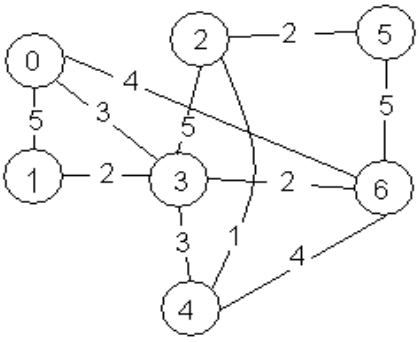
5


0
4
5
2
3
4
2
5
3
1
1
5

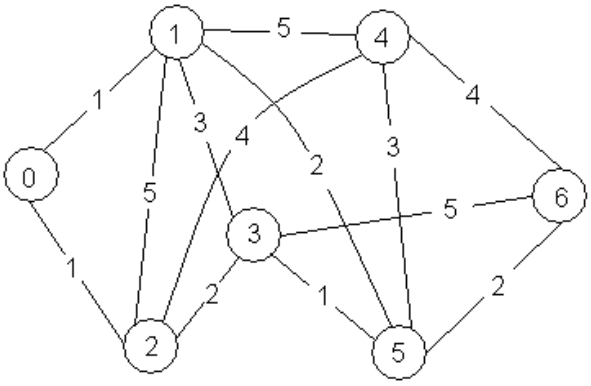
6


0
3
5
2
5
5
3
3
4
2
2
1
3
4
1
5
4

7

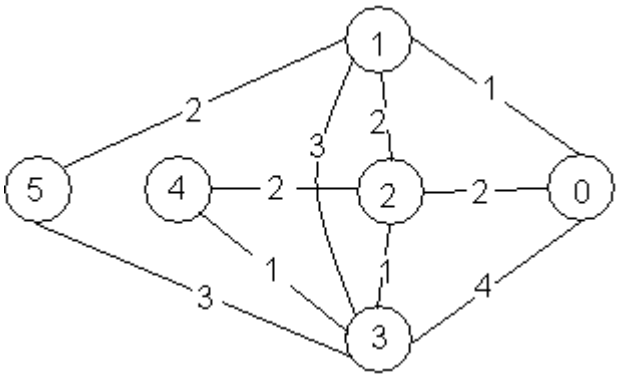

2
2
5
0
4
5
3
5
1
2
3
2
6
3
1
4
4

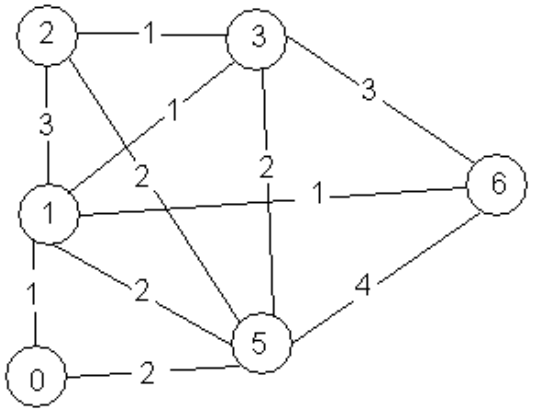
8


1
5
4
1
4
3
4
2
3
0
5
6
5
3
1
2
1
2
5

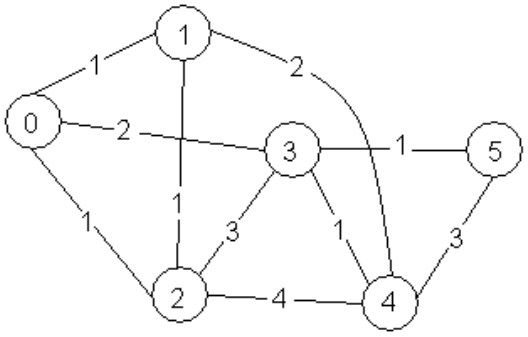
9

10

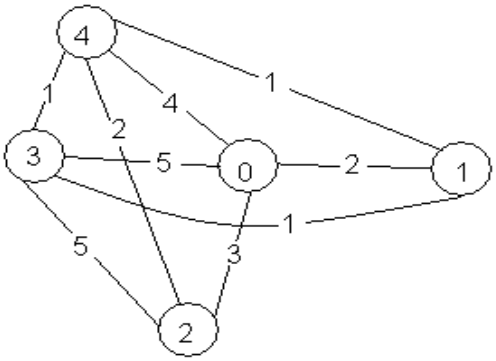


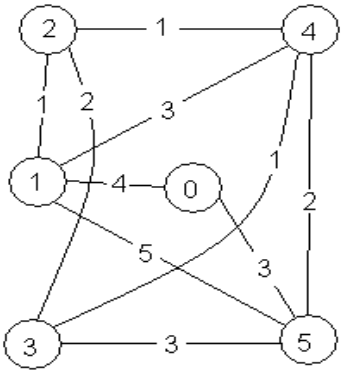

11



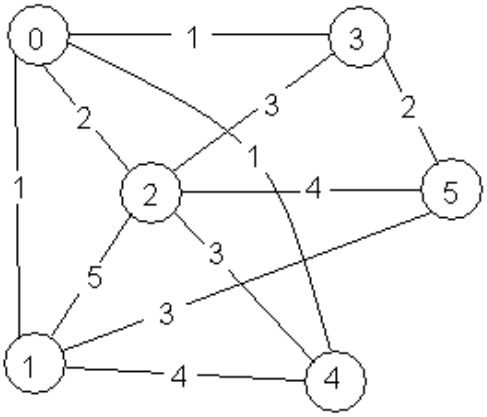

12



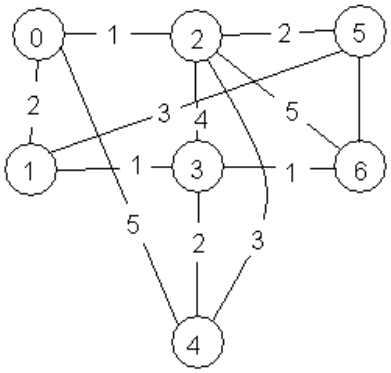

13



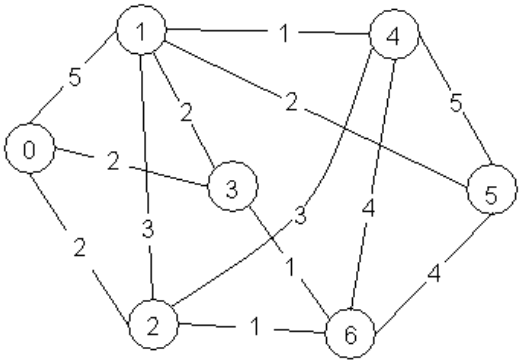

14



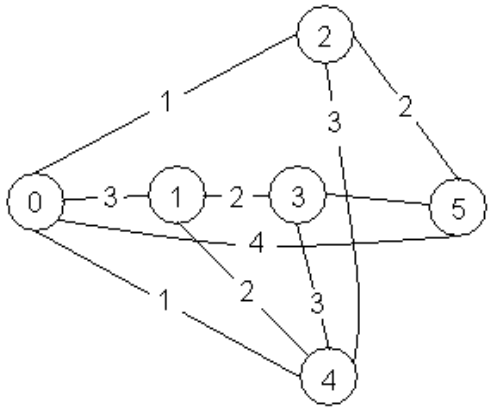

15



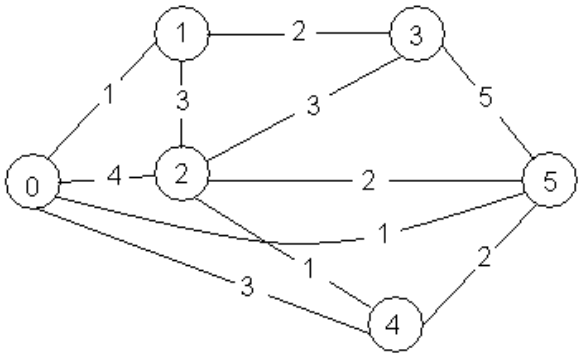

16



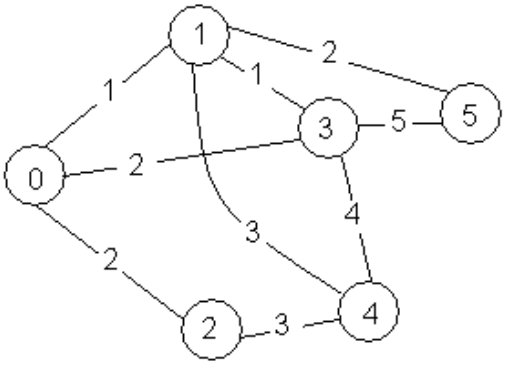
17



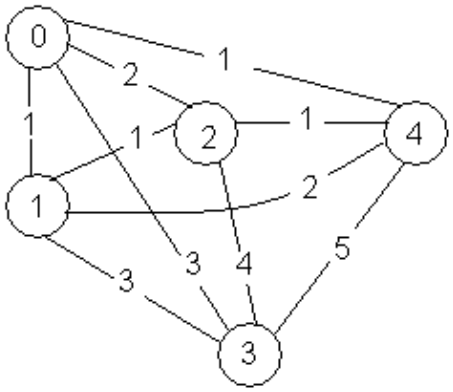

18



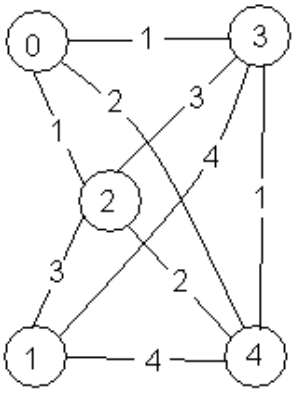

19



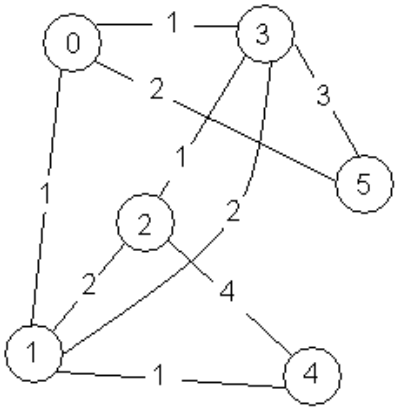

20




21

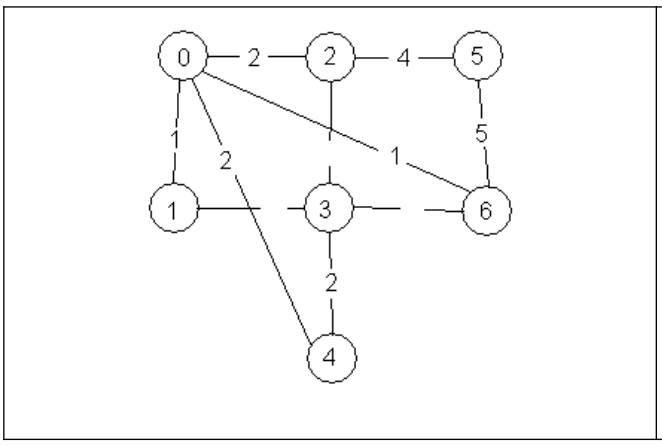


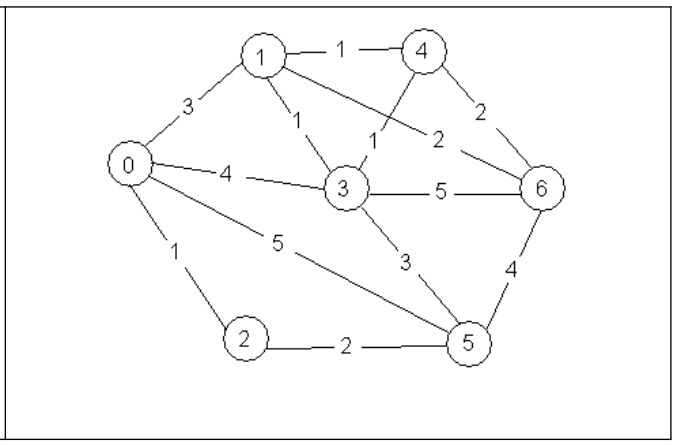

22




23

24




# ЛІТЕРАТУРА


1. Алгоритми та структури даних: конспект лекцій для студентів напряму 6.050103 – Програмна інженерія. Частина 1. Структури даних./Укладачі: О.Д. Воробйова, Л.В. Глазунова. – Одеса: ОНАЗ ім. О.С. Попова, 2017.– 48 с.
2. Алгоритми та структури даних: конспект лекцій для студентів напряму 6.050103 – Програмна інженерія. Частина 2. Алгоритми пошуку, стиснення даних, внутрішнього та зовнішнього сортування, алгоритми на графах./Укладачі: О.Д. Воробйова, Л.В. Глазунова. – Одеса: ОНАЗ ім. О.С. Попова, 2017.– 52 с.
3. Сборник задач по программированию. /Ред. Глазунова Л.В. – Одесса: ОНАС им. А.С. Попова, 2011. – 211 с.
4. Альфред Ахо. Структуры данных и алгоритмы./ [Альфред Ахо, Джон Э. Хопкрофт, Джеффри Д.Ульман] . − М.: Вильямс, 2007. − 400 с.
5. Вирт Н. Алгоритмы и структуры данных/ Вирт Н.−М., 2012. − 272 с.
6. Седжвик Н. Фундаментальные алгоритмы на С++. Части 1-4/ Седжвик Н.− Киев: Диасофт, 2001. − 688 с.
7. Кнут Д. Искусство программирования. Т. 1-4/ Кнут Д.− М.: Вильямс, 2006. − 682 с.
8. Леен Аммерааль, STL для программистов на С++. – М: ДМК, 1999. − 240 с.

# ЗМІСТ