# Лабораторна робота №5. Робота з покажчиками

**Мета**: засвоїти методи роботи з покажчиками в C++.

## Теоретичні відомості

### Поняття покажчика

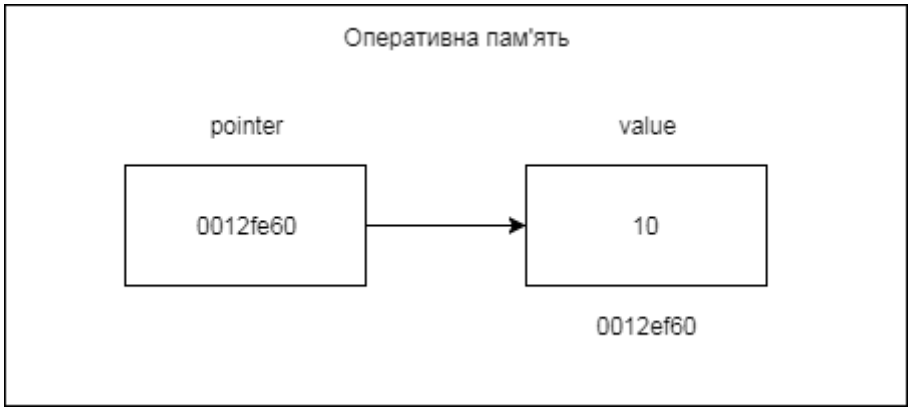
Пам’ять комп’ютера являє собою сукупність комірок для зберігання інформації, кожна з яких має власний номер. Ці номери називаються **адресами.**

Розмір однієї комірки пам’яті становить 1 байт. При оголошенні змінної в пам’яті виділяється ділянка обсягом, відповідним до типу змінної, наприклад 4 байти для типу `int`. Ця ділянка пам’яті пов’язується з іменем змінної. Можна сказати, що адреса змінної – це адреса першої комірки цієї ділянки. Існують спеціальні змінні – **покажчики**, в яких можна зберігати адреси комірок пам’яті, тобто адреси змінних.

Слід звернути увагу на те, що адреса і значення змінної – це зовсім різні поняття. Зазвичай адреси змінних не є важливими, найголовніше – значення цих змінних.

Доступ до змінних зазвичай виконується за її ім’ям. Однак, є альтернативний спосіб доступу – за посиланням, тобто, адресою комірки пам’яті у якій розташовуються дані, що зберігаються у змінній.

Наприклад, нехай оголошена змінна `value` типу `int`, значення котрої зберігається в комірці з адресою 0x0012fe60 та змінна `pointer`, що містить цю адресу. Змінні, що зберігаються адреси інших змінних, називаються покажчиками. Взаємозв’язок між покажчиком та змінною показано на рисунку




Покажчики, як і інші типи даних необхідно оголосити перед використанням. Оскільки значенням покажчика є адреса, за якою

зберігається значення змінної певного типу, під час оголошення слід указати, на який тип посилатиметься покажчик.

Приведемо синтаксис оголошення покажчика:

`<тип>* <ідентифікатор покажчика>;`

Тут `<тип>` - простий тип даних; `<ідентифікатор покажчика>` - рядок символів, що є ім'ям змінної покажчика; символ * означає «вказати на».

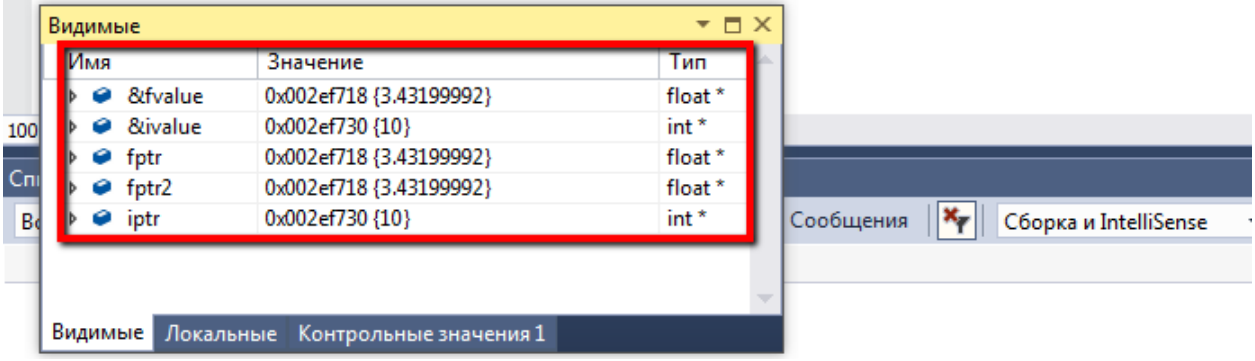
Приклад 1. Оголошення покажчиків.

```
char* cprt;                    // покажчик на символьну змінну
int* iptr;                     //покажчик на цілочислову змінну
float* fptr;                   //покажчик на змінну дійсного
                               //типу
long* lptr1, *lptr2, *lptr3; //серія покажчиків на змінні
                               //long
```

Кожному покажчику потрібно присвоїти значення перед його застосуванням. Покажчик перед використанням ініціалізується адресою змінної або значенням іншого покажчика. Покажчику можна присвоїти значення 0 (NULL). Такий покажчик не адресує жодну змінну. Символьна константа NULL визначена в заголовних файлах `iostream`, `stdio.h`, `stdlib.h`. Значення 0 – це єдине ціле значення, яке можна присвоїти покажчику без приведення його до типу покажчика. Використання неініціалізованого покажчика приводить до помилок часу виконання та перериванню програми.

Приклад 2. Робота з покажчиками.

```
1  //example 4_2.cpp Робота з покажчиками
2  //Приклад 2 до лабораторної роботи №4
3  //Автор: Кудін О.В., 01.10.2017
4  #include "stdafx.h"          // включення заголовних файлів
5  #include <iostream>
6  #include <cmath>
7  using namespace std;         // визначення простору імен
8
9  int main() {
10     char* cptr = "Test string"; // покажчик на рядок символів
11     int ivalue = 10, ivalue2;   // оголошення змінних цілого типу
12     float fvalue = 3.432;       // ініціалізація дійсної змінної
13     int* iptr = &ivalue;        // ініціалізація покажчика на цілий тип
14     float* fptr = &fvalue;      // ініціалізація покажчика посиланням на змінну
15     float* fptr2 = fptr;        // ініціалізація покажчика значенням іншого покажчика
16     return 0;
17 }
```



При наявності ініціалізованого покажчика можливо отримати значення змінної, на яку вказує покажчик. Для цього існує, так звана, операція розіменування.

Приклад 3. Використання операції розіменування

```
1  //example 4_3.cpp Робота з покажчиками
2  //Приклад 3 до лабораторної роботи №4
3  //Автор: Кудін О.В., 01.10.2017
4  #include "stdafx.h"          // включення заголовних файлів
5  #include <iostream>
6  #include <cmath>
7  using namespace std;         // визначення простору імен
8
9  int main() {
10     int n = 10;              // ініціалізація змінної цілого типу
11     int *pn=&n;              // ініціалізація покажчина цілого типу
12     *pn = 5;                 // присвоєння комірці пам'яті, на яку вказує покажчик, цілого значення
13     cout << n;               // вивід значення змінної, яке дорівнює 5 (чому?)
14     return 0;  ≤9 мс прошло
15 }
```

Видимые

| Имя | Значение | Тип |
|---|---|---|
| &n | 0x0023f9b0 {5} | int * |
| *pn | 5 | int |
| n | 5 | int |
| pn | 0x0023f9b0 {5} | int * |

Видимые | Локальные

## Покажчик на тип void

У мовах C\C++ існує покажчик, який може вказувати на будь-який тип даних. Він називається покажчиком типу void і його оголошують так:

void* <ідентифікатор>;

Змінній типу void* можна присвоїти покажчик на будь-який тип, і void-покажчик можна присвоїти покажчику будь-якого типу.

Покажчик на void не можна розіменувати, оскільки він містить адресу пам’яті для невідомого типу даних, розмір якого невідомий компілятору.

Найчастіше void-покажчик використовується для передачі їх у функції, котрі працюють незалежно від типу даних, на які вказує покажчик.

## Поняття посилання

Посилання є псевдонімом змінної, тобто її альтернативним іменем і для нього не резервується місце в оперативній пам’яті. Основне застосування посилання полягає в передачі аргументів у функцію та повернення значень у програму. Синтаксис оголошення посилання застосовує символ &, який записують після типу змінної:

`<тип>& <ідентифікатор посилання>`

Посилання слід ініціалізувати ім’ям змінної. Це здійснюється під час оголошення посилання або виклику функції, в яку передаються посилання як аргументи. Використання посилання після його ініціалізації дає той самий результат, що і пряме використання самої змінної. При цьому, типи

посилання та змінної, значенням якої ініціалізуються посилання, мають збігатися.

Приклад 4. Ініціалізація посилань та покажчиків.

```
1  //example 4_4.cpp Покажчики на функцію
2  //Приклад 4 до лабораторної роботи №4
3  //Автор: Кудін О.В., 01.10.2017
4  #include "stdafx.h"
5  #include<iostream>
6
7  using namespace std;
8
9  int main()
10 {
11     int var = 10;        //ініціалізація змінної
12     int &ref = var;      //ініціалізація посилання на змінну
13     int* pint = &var;    //ініціалізація покажчика через змінну
14     int* pint2 = &ref;   //ініціалізація покажчика через посилання
15     return 0;
16 }
```

Видимые

| Имя | Значение | Тип |
|---|---|---|
| &ref | 0x002afb6c {10} | int * |
| &var | 0x002afb6c {10} | int * |
| pint | 0x002afb6c {10} | int * |
| pint2 | 0x002afb6c {10} | int * |
| ref | 10 | int & |
| var | 10 | int |

## Покажчики на функції

Покажчик на функцію містить адресу її в оперативній пам'яті. Ім'я функції – це початкова адреса її коду. Цей покажчик можна передавати іншим функціям, повертати з них, зберігати, присвоювати іншим покажчикам на функції. Основне призначення покажчиків на функції полягає в тому, щоб надати програмісту можливість передавати функції як аргументи під час виклику інших функцій.

Синтаксис оголошення покажчика на функцію:

`<тип> (*<ідентифікатор покажчика>) (<список параметрів>);`

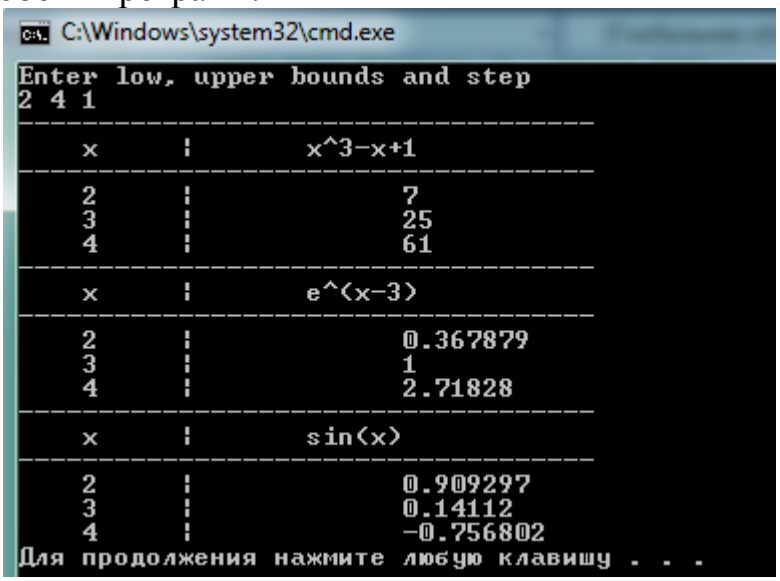
Тут `<ідентифікатор покажчика>` - ім'я покажчика на функцію; `<список параметрів>` - список параметрів, синтаксис оголошення якого збігається із синтаксисом списків параметрів в оголошення функцій; `<тип>` - тип, що його повертає функція.

Приклад 5. Потрібно розробити програму, що буде друкувати значення декількох функцій ($x^3$-$x$+1; $e^{x-3}$; $\sin x$) на відрізку $[a,b]$ з кроком $h$.

```
//example 5_5.cpp Покажчики на функцію
//Приклад 5 до лабораторної роботи №5
//Автор: Кудін О.В., 01.10.2017
#include "stdafx.h"              // включення заголовних файлів
#include <iostream>
#include <cmath>
using namespace std;       // визначення простору імен
```

```
float lower, upper, step;   // межі відрізка та крок
// функція обчислення полінома
float Polynom(float x) {
	return x*x*x - x + 1;
}
// функція обчислення експоненти
float Exponent(float x) {
	return exp(x - 3);
}
// функція друку на екран, використовує покажчик на функцію float(*funct) (float)
void Print(float a, float b, float h, float(*funct) (float), char s[]) {
	// a - нижня межа відрізку, b - верхня межа,
	// h - відстань між точками,
	// funct - покажчик на функцію
	// s - рядок, назва функції
	float x;
	cout << "-----------------------------------" << endl;
	cout << "    x     |       " << s << endl;
	cout << "-----------------------------------" << endl;
	x = a;
	while (x <= b) {
		cout << "    " << x << "     | ";
		cout << "            " << (*funct) (x) << endl;
		x += h;
	}
}
int main() {
	cout << "Enter low, upper bounds and step" << endl;
	cin >> lower >> upper >> step;
	Print(lower, upper, step, Polynom, "x^3-x+1");
	Print(lower, upper, step, Exponent, "e^(x-3)");
	Print(lower, upper, step, sin, "sin(x)");
	return 0;
	}
```

Результат роботи програми:




## Покажчики та посилання як параметри функції

Існують три способи передачі аргументів у функцію:

- як значення;
- як посилання;
- як покажчики.

Функція, в свою чергу, повертає одне значення, використовуючи оператор `return`.

Якщо аргумент функції передається як параметр-значення, то функція створює локальну змінну, в яку записує копію переданого значення. Його модифікація **не впливає** на значення аргументу. Кількість аргументів у функції можу бути довільною, але значення, що повертається, завжди тільки одне.

При виклику функції з аргументами-покажчиками чи аргументами-посиланнями передаються їх адреси в оперативній пам'яті. Тому функція виконує дії над значеннями змінних, а не над їх копіями і **модифікує вихідні значення аргументів.** Отже, в цьому випадку функція має прямий доступ до значень аргументів. Переваги механізму передачі функції параметрів-посилань і параметрів-покажчиків в можливості повертання множини значень у точку її виклику. Синтаксис оголошення заголовків функцій з параметрами-покажчиками та параметрами-посиланнями такий:

`<тип> <ім'я функції> (<тип*> <ідентифікатор>) // параметр-покажчик`

`<тип> <ім'я функції> (<тип&> <ідентифікатор>) // параметр-посилання`

Тут `<тип>` - тип значення, що повертає функція; `<тип*>` - тип, на який посилається покажчик; `<тип&>` - тип посилання; `<ідентифікатор>` - ідентифікатор покажчика чи посилання.

Приклад 6. Реалізація **перевантажених** функцій swap(), що упорядковують два числа за зростанням чи спаданням. Функції міняють місцями два значення, що вводяться з клавіатури.

```
//example 5_6.cpp Покажчики як параметри функції
//Приклад 6 до лабораторної роботи №5
//Автор: Кудін О.В., 01.10.2017
#include "stdafx.h"              // включення заголовних файлів
#include <iostream>

using namespace std;       // визначення простору імен
int n, m;                                 // оголошення змінних
// передача параметрів-посилань
// впорядкування за зростанням
void swap(int &a, int &b) {
	if (a > b) { int tmp = a; a = b; b = tmp; }
}
// передача параметрів-покажчиків
// впорядкування за спаданням
void swap(int *a, int *b) {
	if (*a < *b) { int tmp = *a; *a = *b; *b = tmp; }
}

int main() {
	cout << "Enter int n=";
	cin >> n;
	cout << "Enter int m=";
	cin >> m;
```

```
	swap(&n, &m); // аргументи адреси змінних
	cout << "descending ordering: n=" << n << " m=" << m << endl;
	swap(n, m);          // аргументи змінні
	cout << "ascending ordering: n=" << n << " m=" << m << endl;
	return 0;
	}
```

## Функції, що повертають покажчики та посилання

Функція може повертати як посилання на будь-який об'єкт, так і покажчик на певний тип.

Приклад 7. Програма здійснює пошук символу в рядку. Для цього розроблено функцію `search()`, яка повертає покажчик на шуканий символ, тобто значення типу `char*`. Як параметри функції використовується покажчик на рядок `char* s` та символ `char c` – ключ пошуку.

```
//example 5_7.cpp Покажчики як параметри функції
//Приклад 7 до лабораторної роботи №5
//Автор: Кудін О.В., 01.10.2017
#include "stdafx.h"               // включення заголовних файлів
#include <iostream>
#include <string.h>
using namespace std;        // визначення простору імен
char* str = "Microsoft Visual Studio";   // ініціалізація покажчика на рядок
char symbol;                             // символ, що шукається
char* search(char* s, char c) {          // функція пошуку
	char* ptr;                           // покажчик на шуканий символ
	bool flag = false;                   // ознака успішності пошуку
	for (int i=0; s[i]!=0;i++)           // поки не кінець рядка
	                                     // переглядати його символи
		if (c == s[i]) {                 // якщо символ знайдено
			ptr = &s[i];                 // визначити покажчик на нього
			flag = true;                 // задати ознако успішності пошуку
			break;                       // вийти з циклу
		}
	if (flag) return ptr;                // повернути знайдений покажчик
	else return NULL;                    // якщо символ не знайдено
}                                        // повернути 0
int main() {
	cout << "Input symbol to search in string \n" << str << "\n" << endl;
	cin >> symbol;                       // ввести шуканий символ
	char* p = search(str, symbol);       // провести пошук
	if (p)                               // якщо символ знайдено
		cout << "number of sybol is " << p - str + 1 << endl; // обчислити його
		                                                      //номер у рядку
	else
		cout << "symbol not found" << endl;
	return 0;
}
```

## Покажчик на покажчик

В мовах C\C++ можна створювати покажчики на іншу покажчики, які містять адреси реальних змінних. Найчастіше покажчики на покажчики використовуються в масивах покажчиків.

Відомо, що доступ до значення змінної в оперативній пам'яті здійснюється за її іменем або за адресою, яка зберігає це значення. Нехай означена змінна `value` типу `int`, яка ініціалізована значенням, наприклад 10.

Також оголошено покажчик `ptr1` на тип `int`, який ініціалізовано адресою змінної `value: int* ptr1=&value`. Маємо перший рівень непрямої адресації, за яким доступ до значення змінної здійснюється через розіменування покажчика `*ptr1`. Означимо покажчик `ptr2` на тип `int`, але ініціалізуємо його адресою покажчика `ptr1: int** ptr2=&ptr1`. Матимемо покажчик, який вказує на інший покажчик.


Синтаксис оголошення покажчика на покажчик такий:

`<тип**> <ідентифікатор>;`

## Виділення та звільнення пам’яті

Одне з найважливіших застосувань покажчиків в C++ - виділення оперативної пам’яті під час виконання програми. Така необхідність може виникнути, наприклад, при роботі з масивами, які користувач вводить з клавіатури але заздалегідь не відомо, скільки елементів може ввести користувач.

Для виділення пам’яті в C++ використовується функція `new`. Її синтаксис:

`<тип>* <ім’я покажчика> =new <тип>`

Приклад 8. Використання функції `new`

```
//example 5_8.cpp Покажчики як параметри функції
//Приклад 8 до лабораторної роботи №5
//Автор: Кудін О.В., 01.10.2017
#include "stdafx.h"
#include <iostream>
int main()
{
	using namespace std;
	int nights = 1001;
	int * pt = new int;         // allocate space for an int
	*pt = 1001;                 // store a value there

	cout << "nights value = ";
	cout << nights << ": location " << &nights << endl;
	cout << "int ";
	cout << "value = " << *pt << ": location = " << pt << endl;

	double * pd = new double;   // allocate space for a double
	*pd = 10000001.0;           // store a double there
```

```
        cout << "double ";
        cout << "value = " << *pd << ": location = " << pd << endl;
        cout << "location of pointer pd: " << &pd << endl;
        cout << "size of pt = " << sizeof(pt);
        cout << ": size of *pt = " << sizeof(*pt) << endl;
        cout << "size of pd = " << sizeof pd;
        cout << ": size of *pd = " << sizeof(*pd) << endl;
        return 0;
}
```

Для звільнення пам'яті, в якій вже немає необхідності використовується функція `delete`. Приклад використання:

```
int* ps =new int;  //виділення пам'яті
…                  //використання пам'яті
delete ps;         //звільнення пам'яті
```

Рекомендується збалансовано використовувати функції `new` і `delete`, в противному випадку можливе явище витоку пам'яті, тобто коли пам'яті виділена, але не використовується. Великий витік пам'яті може призвести до аварійного завершення програми.

Слід пам'ятати, що операція `delete` звільняє тільки пам'ять, яка була виділена функцією `new`. Також, `delete` можна застосувати тільки один раз до однієї ділянки пам'яті.

```
int* ps =new int;  //правильно
delete ps;         //правильно
delete ps;         //помилка
int* b=5;          //правильно
int* pi=&b;        //правильно
delete pi          //помилка
```

## Робота з масивами

Якщо масив створюється під час оголошення – пам'ять для нього розподіляється під час компіляції і не змінюється до закінчення роботи програми. Такий розподіл пам'яті для масиву називається **статичне зв'язування**. За допомогою команди new можливо створити масив під час виконання програми, якщо в цьому є необхідність (наприклад, користувач вказав, що хоче ввести множину даних). Або, можливо вказати розмір масиву під час роботи програми. Такий розподіл пам'яті для масиву називається **динамічне зв'язування**, а самі масиви – **динамічними масивами**.

При статичному зв'язуванні необхідно жорстко задати розмір масиву під час написання програми. При динамічному зв'язуванні програма може прийняти рішення про розмір масиву під час роботи (наприклад, на основі введеної користувачем інформації).

Приклад створення одномірного динамічного масиву:

```
float *ptrarray = new float[10];
```

Видалення виділеної оперативної пам’яті, виділеної для одномірного динамічного масиву, виконується наступним чином:

```
delete[] ptrarray;
```

Приклад створення двомірного динамічного масиву:

```
// ініціалізація двомірного динамічного масиву розміром 2x5:
float **ptrarray = new float*[2]; // дві строки в масиві
for (int count = 0; count < 2; count++)
     ptrarray[count] = new float[5]; // і п’ять стовпців
```

Видалення двомірного динамічного масиву:

```
for (int count = 0; count < 2; count++)
     delete[] ptrarray[count];
```

Приклад 9. Створення та звільнення двомірного динамічного масиву.

```
//example 5_9.cpp Двомірні динамічні масиви
//Приклад 9 до лабораторної роботи №5
//Автор: Кудін О.В., 01.10.2017
#include "stdafx.h"
#include <iostream>
#include <ctime>
#include <iomanip>
using namespace std;

int main(int argc, char* argv[])
{
	srand(time(0));			// генерація випадкових чисел
// динамічне створення двомірного масиву дійсних чисел на десять елементів
	float **ptrarray = new float*[2]; // дві строки масиву
	for (int count = 0; count < 2; count++)
		ptrarray[count] = new float[5]; // п’ять стовпців
// заповнення масиву
	for (int count_row = 0; count_row < 2; count_row++)
		for (int count_column = 0; count_column < 5; count_column++)
			ptrarray[count_row][count_column] = (rand() % 10 + 1) / float((rand()
% 10 + 1)); //заповнення масива випадковими числами від 1 до 10

// вивід масиву
	for (int count_row = 0; count_row < 2; count_row++)
	{
		for (int count_column = 0; count_column < 5; count_column++)
			cout << setw(4) << setprecision(2) <<
ptrarray[count_row][count_column] << "   ";
		cout << endl;
	}
	// !!! видалення двомірного динамічного масиву
	for (int count = 0; count < 2; count++)
		delete[]ptrarray[count];

	return 0;
	}
```

## Завдання до лабораторної роботи

Завдання обираються згідно варіанту.

Програми повинні задовольняти наступним вимогам:

– Передбачити введення даних з клавіатури та виведення результатів на екран консолі.

- Передбачити діалогове меню для вибору певної дії, якщо це потрібно.
- Для кожної окремої дії програми (введення чисел, виведення результату, обчислення) передбачити окрему функцію.
- Обов’язковим є використання коментарів (відповідно до прикладів 1-5) та правил вибору ідентифікаторів в програмі (див. лаб. №2).

Варіант 1.

1. Згенерувати значення елементів одновимірного масиву за допомогою генератора псевдовипадкових чисел, ввівши кількість елементів масиву з клавіатури (див. приклад 9). Знайти мінімальний за значенням елемент і записати його на початок масиву, вивільнивши для нього місце шляхом зсуву елементів масиву вправо.
2. Ввести значення елементів одновимірного масиву, задавши попередньо їх кількість. Визначити кількість входжень до масиву значення кожного з його елементів.
3. Задати квадратну матрицю, ввівши кількість рядків і стовпців з клавіатури. Обчислити слід матриці (слід матриці дорівнює сумі елементів головної діагоналі).
4. Задати квадратну матрицю, ввівши кількість рядків і стовпців з клавіатури. Обчислити суму додатних елементів матриці, що розташовані вище побічної діагоналі.

Варіант 2.

1. Нехай, кожний елемент одномірного масиву розміром *n* зберігає коефіцієнт багаточлена порядку *n*. Причому, коефіцієнт при змінній в степені *k* зберігається в *k*-му елементі масиву. Ввести значення коефіцієнтів двох багаточленів, попередньо задавши їх порядок. Обчислити добуток двох багаточленів.
2. Ввести значення елементів одновимірного масиву, задавши попередньо їх кількість. Обчислити кількість елементів у найдовшій серії. Серія – це послідовність однакових елементів, розташованих підряд.
3. Задати квадратну матрицю, ввівши кількість рядків і стовпців з клавіатури. Обчислити суму елементів, які вище головної і побічної діагоналі.
4. Задати квадратну матрицю, ввівши кількість рядків і стовпців з клавіатури. Обчислити скалярний добуток матриці на введене з клавіатури число.

Варіант 3.

1. Обчислити степені двійки $2^N$ та записати їх в одномірний масив. Число $N$ вводиться з клавіатури. Вивести отриманий масив на екран.
2. З клавіатури вводяться: ціле число $N$($N$>1), перший член $A$ та знаменник геометричної прогресії $D$. Сформувати та вивести масив розміру $N$, який містить перші $N$ членів геометричної прогресії.
3. Задати матрицю, ввівши кількість рядків і стовпців з клавіатури. Вивести послідовності елементів строк або стовпців, які строго монотонно зростають або убувають.
4. Задати матрицю, ввівши кількість рядків і стовпців з клавіатури. Вивести номер строки або стовпця, елементи яких є перестановками (перестановки $N$ елементів містять тільки числа від 1 до $N$ включно).

Варіант 4.

1. З клавіатури вводиться ціле число $N$($N$>2). Сформувати та вивести цілочисельний масив розміром $N$, який містить перші $N$ чисел Фібоначчі.
2. Згенерувати значення елементів одновимірного масиву за допомогою генератора псевдовипадкових чисел, ввівши кількість елементів масиву з клавіатури. Вивести масив на екран у зворотному порядку.
3. Задати матрицю, ввівши кількість рядків і стовпців з клавіатури. Вивести номер строки матриці з мінімальною серед інших строк сумою елементів.
4. Задати квадратну матрицю, ввівши кількість рядків і стовпців з клавіатури. Вивести елементи мінору для введених індексів елемента вихідної матриці.

Варіант 5.

1. З клавіатури вводиться ціле число $N$ і $K$(1=<$K$=<$N$). Згенерувати значення елементів одновимірного масиву розміром $N$ за допомогою генератора псевдовипадкових чисел. Вивести всі елементи масиву, які кратні $K$.
2. З клавіатури вводиться ціле число $N$. Сформувати та вивести дійсний масив розміром $N$, який містить перші $N$ чисел Бернуллі. Числа Бернуллі можна розрахувати як рекурентну послідовність:

$$B_o = 1,$$

$$B_n = \frac{-1}{n+1}\sum_{k=1}^{n}\frac{(n+1)!}{(k+1)!(n-k)!}B_{n-k}.$$

3. Задати матрицю, ввівши кількість рядків і стовпців з клавіатури. Вивести номер строки або стовпця, в яких всі елементи різні.

4. Задати квадратну матрицю, ввівши кількість рядків і стовпців з клавіатури. Обчислити кількість елементів матриці, що не повторюються.

Варіант 6.

1. З клавіатури вводиться ціле число $N$. Згенерувати значення елементів одновимірного масиву розміром $N$ за допомогою генератора псевдовипадкових чисел. Вивести всі елементи масиву, з парними номерами (в порядку зростання номерів). Вивести всі елементи масиву, з непарними номерами (в порядку убування номерів). Умовний оператор не використовувати.
2. Ввести значення елементів одновимірного масиву, задавши попередньо їх кількість. Вивести мінімальний елемент з елементів з парними індексами.
3. Задати матрицю, ввівши кількість рядків і стовпців з клавіатури. Обчислити суму елементів, які менше середнього арифметичного всіх елементів матриці.
4. Задати матрицю, ввівши кількість рядків і стовпців з клавіатури. Обчислити середнє геометричне всіх елементів матриці.

Варіант 7.

1. Ввести значення елементів одновимірного масиву, задавши попередньо їх кількість. Вивести всі послідовності елементів масиву, які монотонно зростають.
2. Ввести значення елементів одновимірного масиву, задавши попередньо їх кількість. Вивести номера тих елементів масиву, які більші свого лівого сусіда і кількість таких елементів.
3. Задати матрицю, ввівши кількість рядків і стовпців з клавіатури. Вивести номери строк та стовпців, суми елементів яких дорівнюють.
4. Задати матрицю, ввівши кількість рядків і стовпців з клавіатури. Вивести номер строки матриці, в якій максимальна кількість простих чисел.

## Контрольні запитання

1. Які види покажчиків підтримує мова C++?
2. Особливості ініціалізації покажчиків.
3. Використання функцій new та delete.
4. Визначення та ініціалізація динамічних масивів.
5. Способи передачі параметрів в функцію.