# МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Національний університет “Львівська політехніка”**



**Інститут післядипломної освіти**

**Кафедра програмного забезпечення систем**

**ЗВІТ**

**Про виконання лабораторної роботи №4**

**«Робота з масивами в С++»**

**з дисципліни «Основи програмування»**

Виконав:

слухач групи ПЗС-11

Гринчук Тарас

Прийняв:

доц. Макар В.М.

« »\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2013 р.

∑ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

ЛЬВІВ – 2013

**Тема роботи**: Робота з масивами в С++.

**Мета роботи:** Навчитися організовувати такі структури даних як масиви та освоїти

основні методи програмування алгоритмів обробки масивів даних засобами мови С++.

## 1. ТЕОРЕТИЧНІ ВІДОМОСТІ

1. Масив являє собою сукупність однотипних змінних, розміщених у послідовно

пронумерованих суміжних комірках пам'яті. Номер елемента масиву задається індексом. *Індексація елементів масиву в С++ починається з 0*. Якщо у масиві N елементів, то перший елемент матиме індекс 0, а останній – індекс (N–1). За способом зв'язування індексів з комірками пам'яті виділяють три категорії масивів: *статичні*, *фіксовані автоматичні* та *динамічні*.

1. Масиви можуть бути одновимірними та багатовимірними. Одновимірні масиви оголошуються таким чином:

*тип ім’я\_масиву[розмір]*

Наприклад, так оголошується статичний масив з іменем numbs, що складається з 10 елементів типу int:

*int numbs[10];*

Двовимірний масив, можна представити у вигляді масиву одновимірних масивів:

*тип ім’я\_масиву [розмір1][розмір2]*

1. Вказівники – це змінні, значеннями яких є адреси пам’яті. Якщо змінна безпосередньо посилається на своє значення, то вказівник посилається на значення змінної не безпосередньо або непрямо. Він тільки володіє значенням пам’яті імені відповідної йому змінної.
2. Масиви тісно пов’язанііз вказівниками. Ім’я масиву є вказівником на його перший елемент. Тому ім'я масиву можна використати в якості бази для зміщення вказівника. Наприклад, вираз:

*numbs[4] = 1*

еквівалентний виразу:

*\*(numbs+4) = 1;*

1. Для заповнення масиву початковими даними достатньо виконати ініціалізацію під час оголошення масиву:

*int numbs[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}*

При цьому не потрібно вказувати розмір масиву – він буде розпізнаний за кількістю введених елементів. Тому можна писати так:

*int numbs[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}*

Ініціалізація двовимірного масиву здійснюється по рядках:

*int numbs[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};*

В результаті перший рядок міститиме числа 1, 2, 3, другий – числа 4, 5, 6, а третій – числа 7,8,9. Для наочності краще при ініціалізації використовувати групування:

*int numbs[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};*

1. У С++ доступ до елементів масиву здійснюється за допомогою спеціального оператора []. В квадратних дужках вказується індекс елемента масиву, а перед ними – його ім'я. Наприклад, присвоєння першому елементу масиву numbs значення 12 виконується так:

*numbs[0] = 12;*

Також присвоєння можна здійснити через вказівник:

*\*numbs = 12;*

**Приклад**. З клавіатури ввести 10 цілих чисел, записавши їх у масив. Роздрукувати введені числа в рядок через два пробіли. Результати показані на рис 1.1.

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

int vect[10]; // оголошення масиву з 10 цілих чисел

for(int i = 0; i < 10; i++)

{ /\* у циклі виводимо підказку про введення чисел \*/

cout<<"Enter a number"<<endl;

cin>>vect[i];

/\* на і-ій ітерації циклу записуємо введене число в і-ий елемент

масиву; змінна циклу має початкове значення 0, оскільки елементи масиву індексуються

починаючи з 0 \*/

}

cout<<"You entered:"<<endl;

for(int i = 0; i < 10; i++)

{ /\* у циклі виводимо всі введені числа, відокремлені двома пробілами \*/

cout<<vect[i]<<" ";

}

\_getch();

}

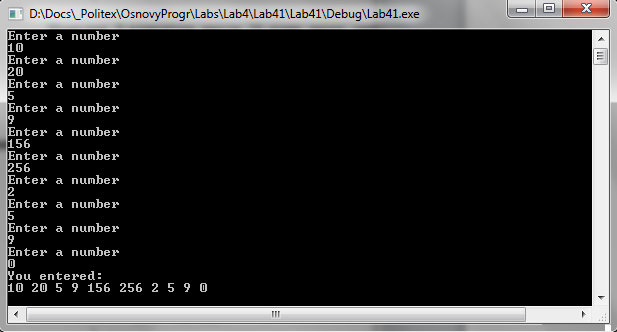


Рис 1.1

**Приклад.** З клавіатури ввести 10 цілих чисел, записавши їх у масив. Роздрукувати введені числа у зворотному порядку. Результати показані на рис 1.2.

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

int vect[10];

for(int i = 0; i < 10; i++)

{

cout<<"Enter a number"<<endl;

cin>>vect[i];

}

cout<<"You entered:"<<endl;

for(int i = 9; i >= 0; i--) /\* елементи масиву «перебираємо» з кінця масиву \*/

{

cout<<vect[i]<<" ";

}

\_getch();

}

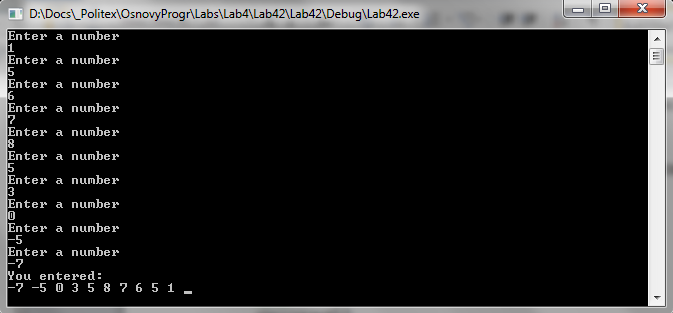


Рис 1.2

1. До масивів застосовний оператор *sizeof*, який повертає розмір масиву у байтах. Наприклад для масиву *char a[5]*, оператор *sizeof(a)*, поверне значення 5. Відповідно, знаючи, що верхній індекс такого масиву буде 4 (5-1), а нижній 0, можемо контролювати, чи індекс не виходить за рамки масиву.

**Приклад**. Ввести з клавіатури 10 символів. Порахувати кількості введених голосних та їх на зразок таблиці. Результати показані на рис 1.3.

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

const char vowels[] = {'a','e','i','o','u','y'}; /\* оголошуємо масив vowels з 6 елементів

типу char та одразу здійснюємо ініціалізацію латинськими голосними літерами; масив

оголошено константним тому, що не передбачається його змінювати протягом виконання

програми \*/

char usersymbols[10]; /\* оголошуємо масив usersymbols, в який будемо записувати 10

символів, введених користувачем з клавіатури \*/

int frequencies[6]; /\* оголошуємо масив frequencies з 6 цілих чисел, який зберігатиме

інформацію про те, скільки разів користувач ввів кожну з голосних літер; тому кількість

елементів масиву frequencies збігається з кількістю елементів масиву vowels \*/

for(int i = 0; i < sizeof(usersymbols)/sizeof(char); i++)

{/\*

в циклі заповнюємо масив usersymbols; вираз sizeof(usersymbols)/sizeof(char) рівний

числу елементів в масиві usersymbols \*/

cout<<"Enter a symbol"<<endl;

cin>>usersymbols[i];

}

for(int i = 0; i < sizeof(vowels)/sizeof(char); i++)

{ /\* у циклі «перебираємо» всі голосні літери, спочатку вважаючи, що кожна з них

зустрілася серед введених користувачем символів 0 разів; і-ий елемент масиву vowels

відповідає і-ому елементу масиву frequencies \*/

frequencies[i]=0;

for(int j = 0; j < sizeof(usersymbols)/sizeof(char); j++)

{ /\* у внутрішньому циклі перевіряємо відповідність кожного введеного символу

і-ій голосній літері; якщо відповідність знайдена, збільшуємо і-ий елемент масиву frequencies

на 1\*/

if(usersymbols[j]==vowels[i])

frequencies[i]++;

}

}

for(int i = 0; i < sizeof(vowels)/sizeof(char); i++)

{ /\* виводимо результати на екран у два стовпці \*/

cout<<vowels[i]<<"\t"<<frequencies[i]<<endl;

}

\_getch();

}

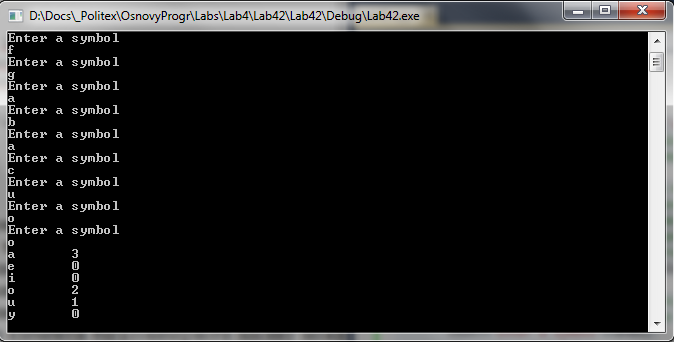


Рис 1.3

1. Багатомірний масив, як і одновимірний, зберігається в послідовних комірках пам'яті, а тому його можна представити як одномірний, пам’ятаючи, що ім'я масиву – це вказівник на його перший елемент.
2. Щоб визначити скільки оперативної пам’яті займає масив, потрібно скористатись оператором *sizeof*, який повертає розмір масиву в байтах.
3. Доступ до елементів багатовимірного масиву можна здійснити через індекси, а також через вказівники. До елементів двовимірного масиву звертаються так:

*ім’я\_масиву [індекс1][індекс2]*

Наприклад:

*matr[4][3] = 10; /\* четвертому елементу п’ятого рядка присвоюється значення 10 \*/*

**Приклад**. З клавіатури ввести квадратну матрицю. Роздрукувати її. Результати показані на рис 1.4.

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

const int N=3;

int matr[N][N];

for(int i = 0; i < N; i++)

for(int j = 0; j < N;j++)

{

cout<<"Enter the "<<(j+1)<<" element of "<<(i+1)<<" row"<<endl; cin>>matr[i][j];

}

for(int i = 0; i < N; i++)

for(int j = 0; j < N;j++)

cout<<matr[i][j]<<"\t";

cout<<endl;

\_getch();

}

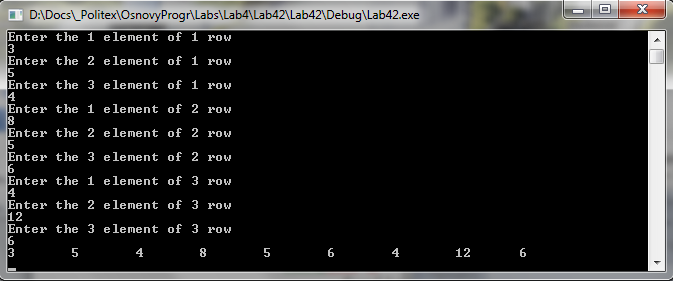
**

Рис 1.4

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

const int N = 4;

int matr[N][N];

for(int i = 0; i < N\*N; i++)

{

/\* квадратна матриця містить N\*N елементів, \*matr є вказівником на перший

елемент першого рядка матриці matr, а змінна і задає зміщення елемента від початку масиву,

оскільки всі елементи розміщені в пам'яті послідовно \*/

cout<<"Enter a number"<<endl;

cin>>\*(\*matr+i);

}

for(int i = 0; i < N; i++)

{

for(int j = 0; j < N;j++)

cout<<matr[i][j]<<"\t";

cout<<endl;

}

\_getch();

}

Оскільки, ім'я масиву – це вказівник на його перший елемент, якщо масив mas має розмірність n\*m, то *mas[i][j]* можна представити як *\*(\*mas+n\*i+j).*

**Приклад**. З клавіатури ввести квадратну матрицю. Роздрукувати її. Результати показані на рис 1.5.

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

const int N = 4;

int matr[N][N];

for(int i = 0; i < N\*N; i++)

{

/\* квадратна матриця містить N\*N елементів, \*matr є вказівником на перший

елемент першого рядка матриці matr, а змінна і задає зміщення елемента від початку масиву,

оскільки всі елементи розміщені в пам'яті послідовно \*/

cout<<"Enter a number"<<endl;

cin>>\*(\*matr+i);

}

for(int i = 0; i < N; i++)

{

for(int j = 0; j < N;j++)

cout<<matr[i][j]<<"\t";

cout<<endl;

}

\_getch();

}

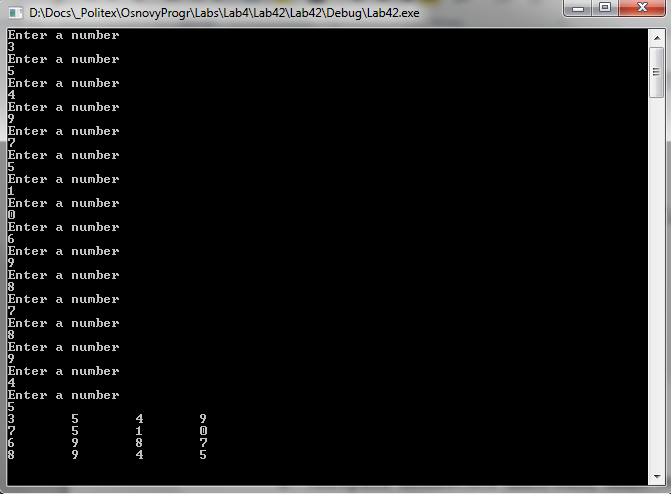


Рис 1.5

***Приклад****.* З клавіатури ввести квадратну матрицю. Роздрукувати саму матрицю та сума елементів якого є максимальною (рядки для зручності нумеруватимемо з 1). Результати показані на рис 1.6.

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

const int N = 4;

int matr[N][N];

for(int i = 0; i < N\*N; i++)

{

cout<<"Enter a number"<<endl;

cin>>\*(\*matr+i);

}

int max = 0;

int maxn = 1;

int sum;

for(int i = 0; i < N; i++)

{

sum = 0;

for(int j = 0; j < N; j++)

sum = sum + matr[i][j];

if(sum > max) {max = sum; maxn = i;}

}

for(int i = 0; i < N\*N; i++)

{

cout<<\*(\*matr+i)<<"\t";

if(!((i+1)%N)) cout<<endl;

}

cout<<"The maximum sum is in the row # "<<(maxn+1);

\_getch();

}

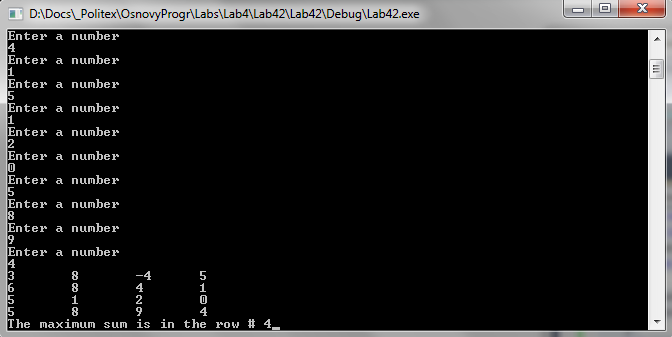


Рис. 1.6

**Приклад**. Ввести з клавіатури квадратну матрицю. Роздрукувати її разом з сумою елементів кожного рядка. Результати показані на рис 1.7.

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

const int N = 4;

int matr[N][N];

for(int i = 0; i < N\*N; i++)

{

cout<<"Enter a number"<<endl;

cin>>\*(\*matr+i);

}

int max = 0;

int maxn = 1;

int sum;

int vect[N]; /\* у vect[i] запишемо суму елементів і-ого рядка матриці matr\*/

for(int i = 0; i < N; i++)

{

sum = 0;

for(int j = 0; j < N; j++)

sum=sum + matr[i][j];

if(sum > max) {max = sum; maxn = i;}

vect[i] = sum;

}

for(int i = 0; i < N; i++)

{

for(int j = 0; j < N;j++)

cout<<matr[i][j]<<"\t";

cout<<"Suma: "<<vect[i]<<endl;

}

\_getch();

}

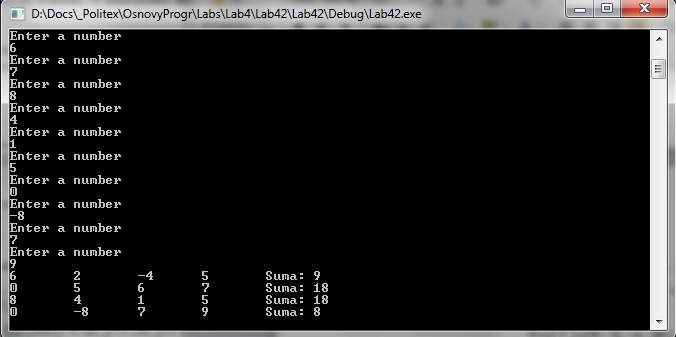


Рис. 1.7

1. У вказівник можна зберегти адресу комірки пам’яті, і зокрема адресу конкретного елементу масиву. Для прикладу адресу першого елемента четвертого рядка масиву *x[5][7]* зберегти у вказівник *ptr* можна такими способами (для зручності будемо розуміти, що ):
   * int \*ptr;

int x[5][7];

ptr = \*x+7\*3; /\* \*x – адреса 1-го елементу, 1-го рядка

\*x+7 – адреса 1-го елементу, 2-го рядка

Відповідно \*x+7\*3 - адреса 1-го елементу, 4-го рядка

\*/

* + int \*ptr;

int x[5][7];

ptr = x[3]; /\* x[0] - адреса 1-го елементу, 1-го рядка

x[1] – адреса 1-го елементу, 2-го рядка

Відповідно x[3] - адреса 1-го елементу, 4-го рядка

\*/

1. Операцію інкремента 6-го елемента масиву array (вважаємо, що перший елемент має номер 1), можна провести такими способами:
   * array[5]++;
   * (\*(array+5))++;

## ХІД РОБОТИ

**Індивідуальне завдання №1 (варіант 5).** Задано масив цілих чисел. Циклічно зсунути його елементи на 2 позиції вправо.

Циклічний зсув передбачає, що старші елементи масиву переходять на місце зсунутих молодших. Оскільки два наймолодших елементи під час зсуву затираються, їх потрібно зберегти у буферний масив. Припустимо ми маємо масив *int a[N]*: нам потрібно в спадаючому циклі пройти елементи з індексами, починаючи з *N-1* і до *0*. Якщо індекс циклу відмінний від 0 та 1, присвоюємо молодшому елементу значення сусіднього старшого. Значення елементів з індексами 0 та 1 - отримуємо з буферного масиву.

Блок-схема для даного алгоритму має вигляд (рис 2.1):

-

+

i<N

i=0

i>=N-2

-

+

b[2-N+i]=a[i]

i++

i=N-1

1

N=10

Ввести a[і]

Рис 2.1

-

+

i>=0

i>1

-

+

a[i]=a[i-2]

i--

i=0

1

a[i]=b[i]

-

+

i<N

i++

Вивести a[і]

Рис 2.1

Програмна реалізація даного алгоритму має вигляд:

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

const int N = 10;

int a[N],b[2],i;

for(i = 0; i < N; i++) {

cout<<"Element #"<<i+1<<": "; //нумерацію почнеио з 1

cin>>a[i];

/\* Якщо ітерація циклу на останніх двох елементах -

першим двом елементам b присвоїмо передостанній та останній елемент

масиву a, відповідно

\*/

if(i>=N-2) b[2-N+i] = a[i];

}

cout<<"\nBefore:\n";

for(i = 0; i < N; i++) cout<<a[i]<<"\t"; //масив до перестановки на екран

//два останні елементи пропускаємо і рухаємось до першого елемента

for(i = N-1; i >=0 ; i--)

/\* Перші два елементи копіюємо з буферного масиву b,

всім іншим присвоюєм значення старших на 2 індекси елементів

\*/

\*(a+i) = (i>1 ? \*(a+i-2) : \*(b+i));

//Аналогічна дія, але за допомогою індексів

//a[i] = (i>1 ? a[i-2] : b[i]);

cout<<"\nAfter:\n";

for(i = 0; i < N; i++) cout<<\*(a+i)<<"\t"; //масив після перестановки на екран

\_getch();

}

Запустимо програму на виконання, зі значеннями елементів масиву: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100. В результаті отримаємо масив: 90, 100, 10, 20, 30, 40, 50, 60, 70, 80 (рис. 2.2).

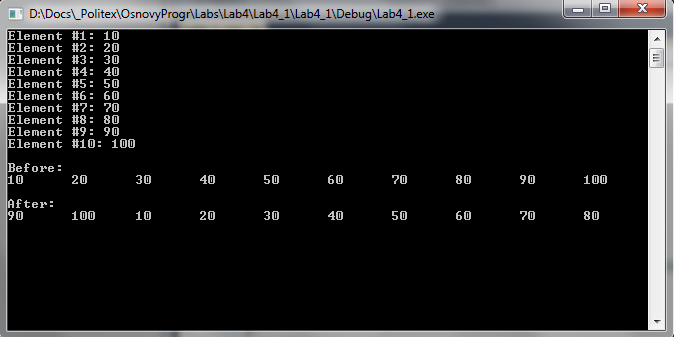


Рис. 2.2

**Індивідуальне завдання №2 (варіант 5).** Знайти найбільший елемент головної діагоналі матриці a[5][5] і вивести на друк весь рядок, де він знаходиться.

Для вводу користувачем матриці, скористаємось вкладеними циклами: у зовніншьому циклі обійдемо рядки матриці, а у внутрішньому – стовбці. Після вводу, виведемо введений масив у вигляді таблиці, для зручності перегляду матриці. Пам’ятаючи, що в С++ елементи масиву зберігаються у пам’яті послідовно рядок за рядком, використаємо для виводу один цикл: for(i=0;i<N\*N;i++), де **N** – константа розміру діагоналі матриці, рівна 5. Доступ до елемента a[i][i] виразимо, як зміщення вказівника першого елементу масиву на **i** позицій: \*(\*a+i). Для того, щоб в кінці кожного рядка таблиці, перевести курсор на нову стрічку, перевіримо чи значення **(i+1)** націло ділиться на **N**, якщо це так – потрібно перевести курсор на новий рядок. Таким чином фрагмент виводу матриці на екран прийме вигляд:

for(i = 0; i < N\*N; i++)

cout<<\*(\*a+i)<<((i+1)%N ? "\t" : "\n");

Щоб знайти максимальний елемент головної діагоналі матриці, потрібно припустити, що елемент **a[0][0]** є максимальним, а потім в циклі пройти всі решта елементи діагоналі та перевірити чи вони є більшими за поточне максимальне значення, у випадку, якщо так: присвоїти значенню максимуму даний елемент і зберегти його індекс. Фрамент коду на С++, з використанням вказівників має виглад:

int imax = 0;

int max = \*\*a; /\* припускаємо, що значення першого елемента є максимальним \*/

for(i = 1; i < N; i++) /\* і=1, оскільки перевіряти, чи a[0][0]>a[0][0] немає сенсу \*/

if(\*(\*a+i\*(N+1)) > max) { /\* Вираз \*(\*a+i\*(N+1)) рівний виразу \*(\*a+i\*N+i) \*/

max = \*(\*a+i\*(N+1));

imax = i;

}

Таким чином у змінній **max** у нас буде збережено максимальне значення елемента головної діагоналі, а у змінній **imax** – його індекс рядка та стовбця, що й виводимо на екран. Для виводу всього рядка, даного елемента, пройдемо у циклі стовбці рядка **imax**, доступ до елемента отримаємо як зміщення вказівника першого елементу рядка **imax** на **j** позицій. Фрагмент коду має вигляд:

for(j = 0; j < N; j++)

cout<<\*(a[imax]+j)<<"\t"; /\* зміщуємо вказівник макс. рядка на j позицій \*/

Блок-схема для даного алгоритму має вигляд (рис 2.3):

-

+

i<N

i=0

j<N

-

+

i++

imax=0

N=5

j=0

Ввести a[і][j]

j++

-

+

i<N

max=a[0][0]

i=1

2

1

3

Рис 2.3

-

+

a[i][i]>max

max=a[i][i]

2

imax=i

i++

1

3

Вивід max

-

+

j<N

j++

Вивід a[іmax][j]

j=0

Рис 2.3

Програмна реалізація даного алгоритму має вигляд:

#include <iostream>

#include <conio.h>

void main() {

using namespace std;

const int N = 5;

int a[N][N],i,j;

for(i = 0; i < N; i++)

for(j = 0; j < N; j++) {

cout<<"Element "<<i+1<<", "<<j+1<<": "; //нумерацію почнемо з 1,1;

cin>>a[i][j];

}

//виведемо матрицю у вигляді таблиці на екран

cout<<"\nMatrix a:\n";

for(i = 0; i < N\*N; i++)

cout<<\*(\*a+i)<<((i+1)%N ? "\t" : "\n"); /\* якщо стрічка не нова: табуляція, інакше перехід на новий рядок \*/

int imax = 0;

int max = \*\*a; //припускаємо, що значення першого елемента є максимальним

for(i = 1; i < N; i++) //і=1, оскільки перевіряти, чи a[0][0]>a[0][0] немає сенсу

if(\*(\*a+i\*(N+1)) > max) { //Вираз \*(\*a+i\*(N+1)) рівний виразу \*(\*a+i\*N+i)

max = \*(\*a+i\*(N+1));

imax = i;

}

cout<<"\nMax element of main diagonal is a ["<<imax+1<<","<<imax+1<<"]: "<<max<<"\nRow:\n";

for(j = 0; j < N; j++)

cout<<\*(a[imax]+j)<<"\t"; //зміщуємо вказівник макс. рядка на j позицій

\_getch();

}

Запустимо програму на виконання та введемо матрицю:

Результат виконання зображено на рис. 2.4.

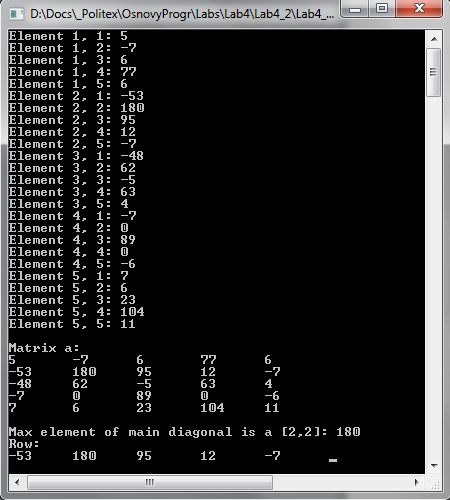


Рис. 2.4

Перевіримо правильність результатів виведених на екран: Головна діагональ матриці містить елементи: {5, 180, -5, 0, 11}. Максимальним числом являється: 180, це елемент, який знаходиться на перетині 2-го рядка та 2-го стовбця матриці – результат вірний. Рядок виведений на екран: {-53, 180, 95, 12, -7} – є 2-им рядком введеної матриці, можна зробити висновок, шо програма працює коректно.

## ВИСНОВКИ

На даній лабораторній роботі я навчився організовувати такі структури даних як масиви та освоїв основні методи програмування алгоритмів обробки масивів даних засобами мови С++. Зокрема навчився працювати з одно- і двовимірними масивами, зчитувати та записувати елементи масивів, як за допомогою індексів, так із використанням вказівників на масиви.