**Лекція 3.** **Покажчики та масиви.**

**Використання масивів**

Опис масивiв у програмі відрізняється від опису простої змінної наявністю після імені квадратних дужок **«[ ]»**, в яких задається кількість елементів масиву (розмірність).

**У мові C++ нумерація елементів масиву починається з 0.**

**<тип> <ім’я> [n];**

**<тип> <ім’я> [n] = {значення};**

**<тип> <ім’я> [  ] = {значення}; /\*** масив відразу, можна не вказувати його розмір \*/

Приклад:

**float m [6];  
float m [6] = {3.4, 4.5, 5.6, 6.7, 8.9, 10.3};  
float m [  ] = {3.45, 4.56, 5.67, 6.78);**

* Надалі кількість елементів змінити неможливо.
* Для того щоб обнулити елементи оголошеного масиву, достатньо ініціювати його перший елемент: **int mas[0]={0};**.
* За замовчуванням, якщо в оголошеному масиві ініціюється тільки декілька перших елементів, то його інші елементи ініціюються нулями. Так, у випадку, коли **float mas[10]= {2.2,34.56};**, останні вісім елементів масиву одержать значення **0**.

#include <iostream.h>

#include <math.h>

int main ( )

{ const int n = 7;

float x[n], y, a(10.5);

int i;

a= 10.5;

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

{

cin >> x[i]; //введення значення

y = a \* x[i] \* x[i] - sin(x[i]);

cout << " x[" <<i<<"] ="<<x[i]<<" y = " << y << endl;

}

system("pause");

return(0);

}

Часто застосовуються багатовимірні масиви. У них позиція елемента визначається записом декількох індексів. Найбільш розповсюджені **двовимірні масиви** або матриці.

Масив задається або списком елементів у тому порядку, и якому вони розташовані у пам’яті, або подається як масив масивів, кожний з яких поміщається в свої фігурні дужки**«{}»**. При оголошенні і одночасному ініціюванні багатовимірних масивів можна опускати кількість індексів тільки першого виміру. Якщо ініціювання не здійснюється під час оголошення масиву, то кількість індексів треба вказувати явно.

**Для здійснення введення-виведення, а також для обробки елементів двовимірного масиву у програмі слід передбачати організацію двох циклів**: один — для завдання значень індексу рядків, другий — індексу стовпців.

#include <iostream.h>

main()

{

const int n = 3, m = 4;

float M [n][m], z = 10;

int i, j;

cout << "\*\*\*\* Vvod matrix " << endl;

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

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

{ cout << " M [" << i << "]" << "[" << j << "]=";

cin >> M [i][j];

M [i][j] += z; // M [i][j]= M [i][j] + z;

}

cout << "\n\n\*\*\*\*\* Rezult matrix: ";

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

{ cout << endl;

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

cout << M [i][j] << " ";}

system("pause");

}

У програмі при описі матриці **float M[n][m];** вказується діапазон змiни двох iндексiв, перший з яких призначений для iндексування рядків **(і)**, другий — для індексування стовпців **(j)**. Введення, обробка і виведення елементів матриці здійснюються за допомогою двох циклів, один з яких є вкладеним в іншій. Це дозволяє при кожному значенні змінної **і**перебирати всі значення змінної **j**.

**Покажчики та масиви.**

**Покажчики** — це змінні, котрі містять адресу пам’яті, розподіленої для об’єкта відповідного типу. При оголошенні змінної-покажчика слід вказати тип даних, адресу яких буде містити змінна, та ім’я покажчика з символом «\*».

Загальний формат опису покажчика має вигляд:

**тип \*ім’я;**

де **тип** — тип значень, на який вказує покажчик;  
**ім’я** — ім’я змінної-покажчика;  
«\*» — операція над типом, що читається «покажчик на тип».

Наприклад:

**int \*рn** – покажчик на ціле значення;  
**float \*pf1, \*pf2;** — два покажчики на дійсні значення.

Покажчики не прив’язують дані до якого-небудь визначеного імені змінної і можуть містити адреси будь-якого неіменованого значення. Існує адресна константа **NULL**, що означає порожню адресу.

Мова C++ налічує лише дві операції, які стосуються адрес змінних, а саме:

**«&»** — **операція взяття адреси** («адреса значення»);

**«\*»** — **операція розіменування** («значення за адресою»).

Операція взяття адреси **«&»** застосовується разом зі змінною і повертає адресу цієї змінної. Операція розіменування «\*» використовується разом з покажчиками і вилучає значення, на яке вказує змінна-покажчик, розташована безпосередньо після символа «\*».

Оголошення покажчиків можна здійснити одним з таких способів:

**<тип> \*ptr;**  
**<тип> \*ptr = <змінна-покажчик>;**  
**<тип> \*ptr = &<ім’я змінної>;.**

Наприклад:  
**int \*ptx, b; float у;** — оголошені змінна-покажчик **ptx** та змінні **b і у**;

**float \*sp = &у;** — покажчику **sp** присвоюється адреса змінної **у**;

**float \*р = sp;** — покажчику **р** присвоюється значення (адреса значення), яке міститься в змінній **sp**, тобто адреса змінної **у**.

При оголошенні покажчиків символ «\*» може знаходитися перед ім’ям покажчика або відразу після оголошення типу покажчика і поширювати свою дію тільки на одну змінну-покажчик, перед якою він записаний:

**long \*pt;   long\*Uk;   int \*ki, x, h;** — оголошення описів.

За потреби для опису покажчика на комірку довільного типу замість ідентифікатора типу записується слово **void**, а саме:

**void \*р, \*pt;** — опис двох покажчиків на довільний тип даних.

**Перед використанням покажчика у програмі його обов’язково необхідно ініціювати**, іншими словами, необхідно присвоїти адресу якого-небудь даного, інакше можуть бути непередбачені результати.

Для одержання доступу до значення змінної, адреса якої зберігається в покажчику, досить у відповідному операторі програми записати ім’я покажчика з символом «\*» — здійснити операцію розіменування.

Розглянемо фрагмент програми з поясненнями:

**int \*р, \*р1;** — оголошені два покажчики на комірку пам’яті типу **int**;

**int х = 12, у = 5, m[7];** — оголошені змінні **х**, **у** і масив **m**, змінні ініційовані;

**р = &у;**     // р (&у); — покажчику **р** присвоєна адреса змінної **у**.

**cout << “Адреса р ” << р << “Значення за цією адресою = ” << \*р; ,**

Виведеться адреса комірки пам’яті, де записана змінна **у** і значення цієї змінної (**тобто 5**).

Використовуючи запис **х = \*р;**, одержимо **х = 5** тому, що **\*р = у = 5;**.

Змінити величину параметра **у** можна так:

**у = 10;**         // \*Р= 10;  
**\*р = \*р+5;**    //у +=5;.

Остання операція означає збільшення значення змінної цілого типу на **5**, тобто **у= 15**.

При ініціюванні покажчиків їм можна присвоювати або адресу об’єкта (змінної), або адресу конкретного місця пам’яті (масиву), або число 0 (нуль), а саме:

**int \*pt = (char \*) 0x00147;** — присвоюється адреса комірки;

**int \*arrpt = new int [10];** — визначається початкова адреса розміщення динамічного масиву;

**char \*р = 0;** — здійснюється ініціювання нулем.

Оскільки покажчики — це спеціальні змінні, то в операціях з іншими покажчиками вони можуть використовуватися без символа «\*», тобто без розкриття посилання, наприклад:

**float \*pt1, \*pt2, х=15, m[5];**

**pt1 = &x;**

**pt2 = pt1;**

**pt1 = m**;         //pt1 = &m[0];

де **m** — ім’я масиву, що розглядається як спеціальний покажчик-константа.

***Приклад***

#include <iostream.h>

int main ( )

{ int x = 10;

int \*px (&x); // int \*px = &x;

cout << "x =" << x << endl;

cout << "\*px =" << \*px << endl;

x \*= 2; //x=x\*2;

cout << "нове значення \*px = " << \*px << endl;\*

px += 2; // \*px=\*px + 2;

cout << "результат \*px, тобто x = " << x << endl;

system("pause"); //затримка екрану

}

Результат виконання програми:  
**х = 10**  
**\*рх = 10**  
**Новое значение \*рх = 20   
Результат \*рх, т. е. х = 22**

Для змінної-покажчика існує своя адреса і тому будуть доцільними записи:

**int \*pt1, \*pt2;**

**pt1 = (int\*) &pt2;** — покажчику **pt1** присвоюється адреса пам’ятi де розташована

змінна **pt2**.

Це має сенс у випадку, коли

**int у, \*pt1, \*pt2 = &у;**

**pt1 = (int\*) &pt2;.**

**Обмеження на застосування операції взяття адреси**:

* не можна визначати адресу літеральної константи (оскільки для неї не виділяється комірка пам’яті), тобто такий запис, як **vp = &345;** — неприпустимий;
* не можна визначати адресу результату арифметичного виразу, тобто запис **vp = &(x + y);** теж неприпустимий.

**Дозволені операції для змінних-покажчиків:**

* операція розіменування «\*»;
* операція взяття адреси «&»;
* операція присвоювання «=»;
* операції інкремент «++» і декремент « –-»;
* операції додавання «+» і віднімання «-»;
* операції відношення (порівняння) покажчиків однакового типу: «==», «!=», «<», «<=», «>», «>=».

У мові C++ масиви і покажчики зв’язані між собою: ***ім’я масиву визначається як покажчик-константа на початковий (нульовий)елемент масиву.*** Так, наприклад, при оголошенні одновимірного масиву у вигляді **int mas [20];** його ім’я **mas** – покажчик на адресу початкового елемента масиву **&mas[0]**.

Існує два способи доступу до елементів масиву:

* з використанням індексу елемента масиву, наприклад, **mas[2]** або **mas[i];**
* з використанням адресного виразу, тобто виразу з покажчиком на масив, наприклад, **\*(mas + 2)** або **\*(mas + і)**.

Ім’я покажчика на масив можна записати так:

**int mas [20];**  
**int \*ptr1;**  
**ptr1 = mas;**      *//ptr1* *= &mas[0];,*

тут вирази **&mas[0] і mas** — еквівалентні.

Оскільки в комп’ютері для масивів завжди є суцільний блок комірок пам’яті, в яких розташовуються їх елементи, то адресу наступного елемента **mas[1]** можна вказати шляхом збільшення покажчика на **1**, а саме:

**р = &mas[0];**

**р++;** *//р=р  + 1;*

Таким чином, адреса **і**-го елемента визначається як **р + і**. При цьому з урахуванням типу масиву і відведеної кількості байтів для кожного його елемента автоматично виконується операція збiльшення адреси, тобто:

**адреса х[і] = адреса х[0] + i\*sizeof (тип);** .

***Для покажчиків, які посилаються на елементи масивів різних типів, результат арифметичних операцій і операцій відношення невизначений.***

До двох покажчиків **р1 і р2**, що вказують на елементи одного масиву, застосовують операції відношення: «==», «!=», «<», «<=», «>», «>=». При цьому значення покажчиків розглядаються як цілі числа, а результат порівняння дорівнює **0** ( »неправда») або **1** («істина»). Так, відношення вигляду **р1<р2** є «істина», якщо **р1** указує на більш ранній елемент, ніж **р2**. Будь-який покажчик можна порівняти з нулем.

В арифметиці з покажчиками можна використовувати адресу неіснуючого «наступного за масивом» елемента. До покажчиків можна додавати або віднімати від них цілу величину.

В обох випадках результатом операції буде покажчик на вихідний тип, значення якого на вказане число елементів більше або менше вихідного. Тобто, якщо до покажчика **р** можна додати деяку цілу величину **n**, а саме: **р + n**, то цей вираз визначає ділянку об’єкта, що займає **n**-не місце після об’єкта, на який вказує **р**, при цьому **n** автоматично збільшується на коефіцієнт, що дорівнює відповідній довжині об’єкта. Наприклад, якщо **int** займає 4 байти, то цей коефіцієнт дорівнює чотирьом.

Допускається також операція віднімання покажчиків, що вказують на елементи одного масиву. Так, якщо **р1 < р2**, то **р2 – р1 + 1** — це число елементів масиву від **р1** до **р2** включно.

***Приклад.*** Обчислити середнє значення додатних елементів одновимірного масиву. перший варіант

*/\* Р671.СРР — вычисление среднего значения положительных элементов массива \*/*

*//---------------- программа без использования указателей*

**#include <iostream.h>**

**main( )**

**{ const int n = 10;**

**float mas[n], s = 0;**

**int i, kol = 0;**

**cout << "Ввод массива " << endl;**

**for(i =0; і < n; i++)     cin >> mas[i];**

**for(i =0; і < n; i++)**

**if (mas[i] > 0)**

**{ s += mas[i];** *//накопление суммы*

**kol++;** **}** *//подсчет положительных елементов*

**cout.precision(3) ;**

**cout << "Средн. арифм. полож. элементов = " << s/kol << endl;**

**system("pause");**   *//задержка экрана вывода резулътата*

**}**

Результати виконання програми:  
**Ввод массива**  
**1.56 -4.78 6.5 7.89 -3.6 9.45 7.4 -8.43 9.3 -10.2**  
**Средн. арифм. полож. элементов = 7.02**

Використовуючи ім’я масиву як покажчик на початок масиву (перший елемент), можна навести другий варіант програми:

**#indude <iostream.h>**

**main ( )**

**{ const int n = 10;**

**float mas[n], s;**

**int i, kol = 0;**

**s = 0;**

**for (i =0; i < n; i++)**

**{ cin >> \*(mas+i);**

**if (\*(mas+i) > 0)**

**{ s += \*(mas+i);**

**kol++; }**

**}**

**cout.precision(3);**

**cout <<"\n Среднее арифм. полож. элементов = " << s/kol << endl;**

**system("pause");**

**}**

Якщо описати покажчик і зв’язати його з масивом (адресувати на початок масиву), то з використанням арифметики покажчиків можна написати третій варіант цiєї програми.

*// использование арифметики указателей*

**#include <iostream.h>**

**main ( )**

**{ const int n = 10;**

**int і, kol(0);**

**float mas[n], s(0);**

**float \*pm = mas;**         *//pm= &mas[0];*

**for (i =0; i < n; i++)**

**{ cout << "Введите" << i << "элемент mas" << endl;**

**cin >> \*pm++;**

**cout << mas[i] << endl;**

**if (mas[i] >0)**

**{ s += mas[i];**

**kol++; }**

**}**

**cout.precision(3);**

**cout << "\n Среднее арифм. полож. элементов = " << s/kol << endl;**

**system("pause");**

**}**

У цій програмі для введення масиву застосований покажчик **\*рm**, а для роботи з масивом — ім’я масиву з індексом.

В останньому випадку використання покажчика **\*рm** призвело б до помилкового результату, оскільки цей покажчик в операціях введення збільшує свою адресу **(рm++)** після введення чергового елемента масиву і надалі вказує на ще не введений елемент.

Четвертий варіантпрограмної реалiзації прикладу:

*/\* использование указателей*

**#include <iostream.h>**

**main ( )**

**{ const int n = 10;**

**float mas[n], s = 0;**

**float \*pm = &mas[0];**  *//pm \*= &mas[0];*

**int i, kol = 0;**

**for (i = 0; i < n; i++)**

**{ cout << "Введите" << i << "элемент mas" << endl;**

**cin >> \*pm;**

**if (\*pm >0)**

**{ s += \*pm;**

**kol++;**

**pm++; }**

**}**

**cout.precision(3);**

**cout << ”\n Среднее арифм. полож. элементов = " << s/kol << endl;**

**system("pause");**

**}**

**Типові помилки програмування**

1. Приклад. Різниця між сьомим елементом масиву та елементом масиву сім. Оскільки нумерація масиву починається з нуля, то сьомий елемент масиву буде мати індекс 6. Елемент масиву 7 насправді є восьмим елементом.

Часто про це забувають, що викликає помилки.

2. Елементам масиву можна присвоювати початкові значення. Наприклад, **float m [6] = {3.4, 4.5, 5.6, 6.7, 8.9, 10.3};**

Якщо початкових значень в списку ініціалізації менше ніж елементів масиву, решта автоматично заповнюється нульовими значеннями. Автоматично нульові значення елементам масиву не присвоюються. Для заповнення масиву нульовими значеннями потрібно це значення присвоїти першому елементу масиву. Якщо потрібна ініціалізація і про це забувають, то це стає джерелом помилки.

3. Завдання в списку ініціалізації більшої кількості елементів ніж розмір масиву є синтаксичною помилкою.

4. Масив може бути оголошений як константа. Наприклад:

**const float m [6];**

Значення такому масиву присвоюють при оголошенні. Присвоювання значень в операторі під час виконання є синтаксичною помилкою.

5. Завершення директиви препроцесора **#include** крапкою з комою – директиви препроцесора не є операторами С++.

6. Посилання на елемент за межами масиву. Ефекти щодо таких посилань залежать від системи.

7. Оголошення

**char string2[20];**

створює символьний масив для 19 елементів, останній елемент - '\**0'** позначає кінець тексту. Оператор

**cin >> string2;**

зчитує рядок з клавіатури до  **string2**.

Недостатній розмір масиву, до якого вводиться символьний рядок, може привести до втрати даних в програмі та до інших помилок.

8. Помилковим є вважати, що елементи локального масиву **static,** в функції отримують нульові значення при кожному виклику функції.

9. Операція \* не поширюється на всі змінні в оголошенні. Кожний покажчик повинен оголошуватися окремо.

10. Разіменування покажчика, який не був правильно ініціалізований або якому не присвоєно значення, що вказує конкретне місце в пам’яті може викликати невиправну помилку виконання або неочікуваним чином змінити дані, що приведе до неправильних результатів.

**Техніка програмування**

1. Визначення розміру масиву через іменовану константу робить програму більш маштабованою.

**const arraySize=10;**

**float m [arraySize];**

**Хороший стиль програмування**

1. Намагайтеся програмувати зрозуміло. Інколи краще пожертвувати більш високої ефективності використання пам’яті або процесорного часу задля більшої зрозумілості програми. Але до цього процесу потрібно підходити зважено, інколи ефективність більш важлива ніж зрозумілість. В цьому випадку потрібно пам’ятати про важливість коментарів.

2. При використанні циклів з масивом індекс масиву не повинен бути меншим за нуль і повинен бути меншим за кількість елементів масиву.

3. В програмі повинна забезпечуватися правильність всіх значень, що вводяться, для унеможливлення впливу помилкової інформації на результати обчислення.

4. Хоч це і не обов’язково, доцільно в імена змінних покажчиків додавати символи Ptr.

5. Покажчики повинні інціалізуватися або при оголошенні або за допомогою оператора присвоювання. Покажчик може отримувати значення NULL, 0 або адресу. Для унеможливлення непередбачуваних результатів покажчикам потрібно присвоювати початкові значення.

6. Існує негласна домовленість, що імена змінних повинні починатися з маленької літери, а слова в середині імені з великої, наприклад, **arraySize.**

**7.** Імена змінних повинні бути короткими та наочними. Уникайте імен, які не мають сенсу.

***Контрольні запитання для самоперевірки***.

1. Яким чином задається масив?
2. Як здійснюється введення-виведення двовимірних масивів?
3. Що таке покажчик і як він пов’язаний з адресом?
4. Які існують обмеження на операції взяття адреси?
5. Які операції можуть застосовуватися до змінних-покажчиків?
6. Які типові помилки програмування виникають при роботі з масивами?

**Для самостійного вивчення** *(4 години)*: Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

**Рекомендована література**

1. Ковалюк Т. В. Алгоритмізація та програмування: Підручник. — Львів: «Магнолія 2006», 2013. — 400 с., ил.
2. Павловская Т.А. С/С++. Программирование на языке высокого уровня. СПб.: Питер, 2003. – 461 с. URL: <http://www.ph4s.ru/bookprogramir_1.html>
3. Вступ до програмування мовою С++. Організація обчислень: навч. посіб. / Ю. А. Бєлов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставовський. – К. : Видавничо-поліграфічний центр "Київський університет", 2012. – 175 с. с.: іл. ISBN (укр.) . URL: <http://csc.knu.ua/uk/library/books/belov-24.pdf>
4. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..

5. Дейтел Х., Дейтел П. Основы программирования на С++. – М.: Бином, 1999. – 1024 с. URL: <http://ijevanlib.ysu.am/wp-content/uploads/2018/03/deytel.pdf>