**Лекція 5**

**Масиви**

**План**

1. Одновимірні масиви.
2. Створення вказівника на масив.
3. Передача одновимірного масиву у функцію.
4. Двовимірні масиви
5. Багатовимірні масиви
6. Індексація вказівників
7. Ініціалізація масиву

**Масив**(array) — це сукупність змінних, що мають однаковий тип й об'єднаних під одним ім'ям. Доступ до окремого елемента масиву здійснюється за допомогою індексу. Відповідно до правил мови C/C++ всі масиви складаються із суміжних комірок пам'яті. Молодша адреса відповідає першому елементу масиву, а старший - останньому. Масиви можуть бути одновимірними й багатовимірними. Найпоширенішим масивом є рядок, що завершується нульовим байтом. Вона являє собою звичайний масив символів, останнім елементом якого є нульовий байт.

Масиви й вказівники тісно зв'язані між собою. Важко описувати масиви, не згадуючи вказівники, і навпаки.

**Одновимірні масиви**

Оголошення одновимірного масиву виглядає в такий спосіб.

*тип ім'я\_змінної[розмір]*

Як й інший змінній, масив повинен оголошуватися явно, щоб компілятор міг виділити пам'ять для нього. Тут *тип* повідомляє базовий тип масиву, тобто тип його елементів, а *розмір* визначає, скільки елементів утримується в масиві. От як виглядає оголошення масиву з ім'ям **balance,** щомає тип **double** і складається з **100** елементів.

double balance[100] ;

Доступ до елемента масиву здійснюється за допомогою імені масиву й індексу. Для цього індекс елемента вказується у квадратних дужках після імені масиву. Наприклад, оператор, наведений нижче, привласнює третьому елементу масиву **balance** значення 12.23.

balance[3] = 12.23;

Індекс першого елемента будь-якого масиву в мові C/C++ дорівнює нулю. Отже, оператор

char p[10];

повідомляє масив символів, що складає з 10 елементів — від **р[0]** до **р[9].** Наступна програма заповнює цілочисельний масив числами від 0 до 6.

#include <iostream>

#include "windows.h"

using namespace std;

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int ABC[6];

for (int i = 0; i < 6; i++)

ABC[i] = i;

for (int i = 0; i < 6; i++)

{

cout << "Елемент ABC[" << i << "]=";

cout << ABC[i] << endl;

}

cout << endl;

system("pause");

return 0;

} Обсяг пам'яті, необхідний для зберігання масиву, залежить від його типу й розміру. Розмір одновимірного масиву в байтах обчислюється по формулі:

*кількість\_байтів* = **sizeof***(базовий\_тип) \* кількість\_елементів*

У мові C/C++ не передбачена перевірка виходу індексу масиву за межі припустимого діапазону. Іншими словами, під час виконання програми можна помилково вийти за межі пам'яті, відведеної для масиву, і записати дані в сусідні осередки, у яких можуть зберігатися інші змінні й навіть програмний код. Відповідальність за запобігання подібних помилок лежить на програмістові. Наприклад, фрагмент програми, наведений нижче, буде скомпільований без помилок, однак під час виконання програми індекс масиву вийде за межі припустимого діапазону.

int count[10], і;

/\* Вихід індексу масиву за межі припустимого діапазону \*/

for(i=0; i<100; i++) count[i] = і;

Власне кажучи, одновимірний масив являє собою список змінних, що мають однаковий тип і зберігаються в суміжних комірках пам'яті в порядку зростання їхніх індексів. На мал. 1 показано, як зберігається в пам'яті масив а, що починається з адреси 1000 й оголошений за допомогою оператора

char а [7];

Елемент а[0] а[1] а[2] а[3] а[4] а[5] а[6]

Адреса 1000 1001 1002 1003 1004 1005 1006

*Рис. 1. Масив із семи символів, що починається з адреси 1000*

**Створення вказівника на масив**

Ім'я масиву є вказівником на перший його елемент. Допустимо, масив оголошений за допомогою оператора

int sample [10];

Вказівником на його перший елемент при цьому є ім'я **sample**. Таким чином, у наступному фрагменті вказівнику привласнюється адреса першого елемента масиву **sample**.

int \*р;

int sample[10] ;

р = sample;

Адресу першого елемента масиву можна також обчислити за допомогою оператора &. Наприклад, вираження **sample** й **&sample[0]** еквівалентні. Однак у професійно написаних програмах мовою C/C++ ви ніколи не зустрінете вираження **&sample[0].**

**Передача одновимірного масиву у функцію**

У мові C/C++ весь масив не можна передати як аргумент функції. Замість цього можна передати вказівник на масив, тобто ім'я масиву без індексів. Наведений нижче фрагмент програми передає у функцію **func1()** індекс **і**.

int main(void)

**{**

int і[10];

func1(i) ;

.

.

**}**

Якщо аргументом функції є одновимірний масив, її формальний параметр можна оголосити трьома способами: як вказівник, як масив фіксованого розміру і як масив невизначеного розміру. Наприклад, щоб надати функції **funс1()** доступ до масиву **і**, можна використати три варіанти. По-перше, у якості її аргументу можна оголосити вказівник.

void fund(int \*x) /\* Вказівник \*/

**{**

**.**

**.**

**.**

**}**

По-друге, можна передати вказівник на масив фіксованого розміру.

void fund(int x[10]) /\* Масив фіксованого розміру \*/

**{**

**.**

**.**

**.**

**}**

І, нарешті, можна використати масив невизначеного розміру.

void funс1(int x[ ]) /\* Масив невизначеного розміру \*/

**{**

**.**

**.**

**.**

**}**

Ці три оголошення еквівалентні один одному, оскільки їхній зміст зовсім однаковий: у функцію передається вказівник на цілочисельну змінну. В першому випадку дійсно використовується вказівник. У другому застосовується стандартне оголошення масиву. В останньому варіанті оголошується, що у функцію буде переданий цілочисельний масив невизначеної довжини. Розмір масиву, переданого у функцію, не має ніякого значення, оскільки перевірка виходу індексу за межі припустимого діапазону в мові C/C++ не передбачена. При вказівці розміру масиву можна написати будь-яке число - це нічого не змінить, оскільки у функцію буде переданий не масив, що складається з 32 елементів, а лише вказівник на його перший елемент.

void func1(int x[32]) /\* Масив фіксованого розміру \*/

**{**

**.**

**.**

**.**

**}**

**Двовимірні масиви**

У мові C/C++ передбачені багатовимірні масиви. Найпростішим з них є двовимірний. Власне кажучи, двовимірний масив - це масив одновимірних масивів. Оголошення двовимірного масиву **d**, що складає з 10 рядків й 20 стовпців, виглядає в такий спосіб.

int d[10][20];

Повідомляючи двовимірні масиви, варто проявляти обережність. У деяких мовах програмування розмірності масивів відокремлюються комами. На відміну від них, у мові C/C++ розмірності масиву беруться у квадратні дужки. Звертання до елемента двовимірного масиву виглядає так:

d[l][2]

Урахуйте також, що відношення "більше" й "менше" між рядками розуміються в лексикографічному змісті. Наприклад, значення **stremp**("А","Я") дорівнює —31 (тобто рядок "А" менше рядка "Я"), а значення **stremp("Я","A")** дорівнює 31. Зверніть увагу на те, що значення s**tremp** (**"АЯ"**, **"ЯА"**) знову дорівнює —31, іншими словами, функція **stremp** повертає різниці між ASCII-кодами перших незбіжних між собою символів. Властиво, саме так упорядковуються слова за абеткою.

Наступна програма заповнює двовимірний масив числами від 1 до 19 і виводить їх на екран по рядках.

#include <stdio.h>

int main(void)

{

int t, і, num**[**3][ 4 ];

for(t=0; t<3,- ++t)

for(i=0; і<4; ++і)

num[t][i] = (t\*4)+i+l;

/\* Вивід на екран \*/

for(t=0; t<3; + + t) {

for(i=0; і<4; ++і)

printf("%3d ", num[t][i]);

printf("\n");

}

return 0;

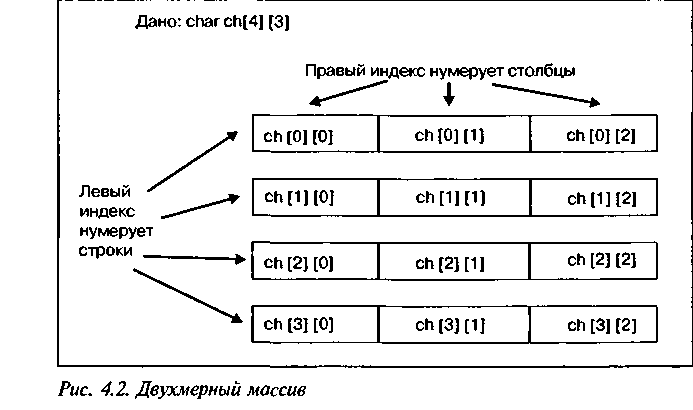
}

У даному прикладі елемент **num[0] [0]** дорівнює **1, num[0] [l]** дорівнює 2, **num[0] [2]** дорівнює 3 і т.д. Значення елемента **num[2] [3]** дорівнює 12. Масив **num** можна зобразити в такий спосіб.

**num[t][і]**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** |
| **0** | **1** | **2** | **3** | **4** |
| **1** | **5** | **6** | **7** | **8** |
| **2** | **8** | **10** | **11** | **12** |

Двомірний масив зберігається у вигляді матриці, у якій перший індекс задає номер рядка, а другий - номер стовпця. Таким чином, при обході елементів у порядку їхнього розміщення в пам'яті правий індекс змінюється швидше, ніж лівий. Графічна схема розміщення двомірного масиву в пам'яті показана па мал. 4.2.



Обсяг пам'яті, займаний двовимірним масивом, виражений у байтах, задається наступною формулою:

*кількість\_байтів=кількість\_рядків\**

*кількість\_стовпців\*****sizeof****(базовий\_тип)*

Вважаючи, що ціле число займає 4 байт, можна обчислити, що для зберігання масиву, що складає з 10 рядків й 5 стовпців, необхідно 10\*5\*4=200 байт.

Якщо двовимірний масив використовується як аргумент функції, то в неї передається тільки вказівник на його перший елемент. Однак при цьому необхідно вказати, принаймні, кількість стовпців. (Можна, зрозуміло, задати й кількість рядків, але це не обов'язково.) Кількість стовпців необхідно компіляторові для того, щоб правильно обчислити адресу елемента масиву усередині функції, а для цього повинна бути відома довжина рядка. Наприклад, функція, що одержує як аргумент двовимірний масив, що складається з 10 рядків й 10 стовпців, може виглядати так.

void func1(int x[ ][10])

**{**

**.**

**.**

**.**

}

Компілятор повинен знати кількість стовпців, інакше він не зможе правильно обчислювати вираження, подібні наступному:

х[2][4]

Якби довжина рядка була невідома, компілятор не знайшов би початок третього рядка.

Розглянемо коротку програму, у якій двовимірний масив використається для зберігання оцінок, отриманих студентами. Передбачається, що вчитель викладає в трьох групах, у яких учаться не більше 30 студентів. Зверніть увагу на те, як відбувається обхід у масиві **grade** у кожній функції.

/\* Проста база даних, що містить оцінки студентів. \*/

#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

#define CLASSES 3

#define GRADES 30

int grade[CLASSES][GRADES];

void enter\_grades(void);

int get\_grade(int num);

void disp\_grades(int g[ ][GRADES]);

int main(void)

{

char ch, str[80];

for(;;) {

do {

printf("(В)від оцінок \n");

printf("(Д)рук оцінок\n");

printf("(К)інець\п");

gets(str);

ch = toupper(\*str);

} while(ch!='B' && ch!='П' && ch!='K');

switch(ch) {

case 'B':

enter\_grades();

break;

case 'П':

disp\_grades(grade);

break;

case 'K':

exit(O);

}

}

return 0;

}

/\* Введення оцінок. \*/

void enter\_grades(void)

{

int t, і;

for(t=0; t<CLASSES; t++) {

printf("Class # %d:\n", t+1);

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

grade[t][i] = get\_grade(i);

• }

}

/\* Зчитування оцінки. \*/

int get\_grade(int num)

{

char s[80];

printf("Введіть оцінку студента # %d:\n", num+1);

gets(s);

return(atoi(s));

}

/\* Вивід оцінок. \*/

void disp\_grades(int g[ ][GRADES])

{

int t, i;

for(t=0; t<CLASSES; ++t) {

printf("Група # %d:\n", t+1);

for(i=0; i<GRADES; ++і)

printf("Студент #%d is %d\n", i+1, g[t][i]);

}

}

**Багатовимірні масиви**

У мові C/C++ масиви можуть мати більше двох розмірностей. Максимально припустима кількість розмірностей задасться компілятором. Загальний вид оголошення багатовимірного масиву такий.

*тип ім'я*[*Розмір1*] [*Розмір2*][*Розмір3*]... [*Розмір4*]

Масиви, що мають більше трьох розмірностей, використовуються рідко, оскільки вони займають занадто великий обсяг пам'яті. Наприклад, чотиривимірний масив символів розмірністю 10x6x9x4 займає 2160 байт. Якби масив містив двохбітові цілі числа, треба було б 4320 байт. Якби елементи масиву мали тип **double** і займали б 8 байт кожний, то для масиву треба було б 17280 байт. Розмір пам'яті, виділеної для масиву, экспоненціально зростає зі збільшенням кількості розмірностей, наприклад, якщо до попереднього масиву додати п'ятий вимір, у якому розташовується 10 елементів, то для нього знадобився б 172800 байт.

При звертанні до багатовимірних масивів компілятор обчислює кожен індекс. Отже, доступ до елементів багатовимірного масиву відбувається значно повільніше, ніж до елементів одновимірного масиву.

Передаючи багатовимірний масив у функції, варто вказувати всі розміри, крім першого. Наприклад, якщо масив **m** оголошений оператором

int m[4][3][6][5];

то функція **func1(),** щоодержує його як аргумент, може виглядати так:

void func1 (int d[ ][3][6][5])

**{**

**.**

**.**

**.**

**}**

Зрозуміло, при бажанні можна вказати й перший розмір, але це не обов'язково

**Індексація вказівників**

У мові C/C++ вказівники й масиви тісно зв'язані один з одним. Як відомо, ім'я масиву без індексу - це вказівник на його перший елемент. Розглянемо, наприклад, що випливає масив.

char p[10];

Наступні два вираження абсолютно ідентичні.

p

&р[0]

З іншого боку, вираження р == &р[0]

є істинним, оскільки адреса першого елемента масиву збігається з адресою всього масиву.

Як вказувалося раніше, ім'я масиву являє собою вказівник. І, навпаки, вказівник можна індексувати, як масив. Розглянемо наступний фрагмент програми.

int \*p, і [10] ;

p = i;

р[5] =100; /\* Присвоювання за допомогою індексу \*/

\*(р+5) = 100; /\* Присвоювання за допомогою адресної арифметики\*/

Обидва оператори присвоєння заносять число 100 у шостий елемент масиву **і**. Перший з них індексує вказівник **р**, а другий використовує адресну арифметику. В обох випадках результат однаковий.

Цей спосіб можна застосовувати й до багатовимірних масивів. Наприклад, якщо змінна **а** — це вказівник на цілочисельний масив, що складається з 10 рядків й 10 стовпців, те наступні два оператори будуть еквівалентними.

**a**

&а[0][0]

Більше того, доступ до елемента, що стоїть на перетині першого рядка й п'ятого стовпця, можна одержати двома способами: або індексуючи масив— **а[0] [4],** або використовуючи вказівник— **\*((int** \*)а+4). Аналогічно елемент, що стоїть в другому рядку й третьому стовпці, є значенням виражень **а[1] [2]** й **\*((int \*)а+12).** Для двовимірного масиву справедлива наступна формула:

**a[j] [k]** =\* *((базовий\_тип* \*)а **+** (і \* *довжина\_рядка)* ***+*** к)

Правила адресної арифметики вимагають приведення типу вказівника на масив до його базового типу. Звертання до елементів масиву за допомогою вказівників використовується досить широко, оскільки операції адресної арифметики виконуються швидше, ніж індексація.

Двомірний масив можна представити за допомогою вказівника на масив одномірних масивів. Отже, використовуючи окремий вказівник, можна звертатися до елементів, що стоять у рядках двовимірного масиву. Цей спосіб ілюструється наступною функцією. Вона виводить на друк вміст заданого рядка глобального цілочисельного масиву **num.**

int num[10][10];

void pr\_row(int j)

**{**

int \*p, t;

p = (int \*) &num[j][0]; /\* Одержати адресу першого елемента у рядку з індексом j \*/

for(t=0; t<10; ++t) printf("%d ", \*(p+t));

}

Цю функцію можна узагальнити, включивши до списку аргументів індекс рядка, її довжину й вказівник на перший елемент масиву.

void pr\_row(int j, int row\_dimension, int \*p)

**{**

int t ;

р = р + (j \* row\_dimension);

for(t=0; t<row\_dimension; ++t)

printf("%d ", \*(p+t));

}

.

.

.

void f(void)

{

int num[10][10];

pr\_row(0, 10, (int \*) num); /\* Вивід першого рядка \*/

}

Цей спосіб можна застосовувати й до масивів більше високої розмірності. Наприклад, тривимірний масив можна звести до вказівника на двомірний масив, що, у свою чергу, можна представити за допомогою вказівника на одномірний масив. У загальному випадку n-вимірний масив можна звести до вказівника на (n-1)-вимірний масив і т.д. Процес завершується одержанням вказівників на одновимірний масив.

**Ініціалізація масиву**

У мові C/C++ допускається ініціалізація масивів при їхньому оголошенні. Загальний вид ініціалізації масиву не відрізняється від ініціалізації звичайних змінних.

*тип\_масиву ім'я\_масиву*[*розмір1*]*…*[*розмірN*] *= {список\_значень}*

*Список\_значень* являє собою список констант, розділених комами. Тип констант повинен бути сумісним з *типом масиву.* Перша константа приcвоюється першому елементу масиву, друга - другому й т.д. Зверніть увагу на те, що після фігурної дужки, що закривається, } обов'язково повинна стояти крапка з комою.

Розглянемо приклад, у якому цілочисельний масив, що складається з 10 елементів, ініціалізується числами від 1 до 10.

int i[10] = {1, 2, 3, 4, 5, б, 7, 8, 9, 10};

Тут елементу **і** [0] привласнюється значення 1, а елементу **і** [9] — число 10. Символьні масиви можна ініціалізувати рядковими константами:

char *ім'я\_масиву[розмір]=" рядок";*

Багатовимірні масиви ініціалізуються аналогічно. Розглянемо приклад ініціалізації масиву **str** числами від 1 до 10 й їхніми квадратами.

int sqrs[10][2] = {

1, 1,

2, 4,

3, 9,

4, 16,

5, 25,

6, 36,

7, 49,

8, 64,

9, 81,

10, 100

**};**

Ініціалізуючи багатовимірний масив, можна брати елементи списку у фігурні дужки. Цей спосіб називається *субагрегатним групованням* (subaggregate grouping). Haприклад попередню ініціалізацію можна переписати наступним чином.

int sqrs[10][2] = {

{1, 1},

{2, 4},

{3, 9},

{4, 16},

{5, 25;

{6, 36}.

{7, 49}

{8, 64},

{9, 81},

{10, 100}

};

Якщо при групованні даних зазначені не всі початкові значення, що залишилися елементи групи автоматично заповнюються кулями.

**Ініціалізація безрозмірного масиву**

Уявіть собі ініціалізацію таблиці повідомлень про помилки, кожне з яких зберігається в одномірному масиві.

char el [12] = "Помилка при читанні\n";

char е2[13] = "Помилка при записі\n";

char ез[18] = "Неможливо відкрити файл\n";

Щоб задати правильний розмір масивів, довелося б підраховувати точну кількість символів у кожному повідомленні. На щастя, компілятор може сам визначити розмір масиву. Якщо в операторі ініціалізації не зазначений розмір масиву, компілятор автоматично створить масив, що вміщає всі початкові значення. Такий масив називається *безрозмірним* (unsized array). Використовуючи цей підхід, можна переписати попередній фрагмент програми інакше.

char el[ ] = "Помилка при читанні\n";

char е2[ ] = "Помилка при записі\n" ;

char е3[ ] = "Неможливо відкрити файл\n";

У цьому випадку оператор

printf("Довжина рядка \"%s\" дорівнює %d\n", е2, sizeof e2);

виведе на екран повідомлення

*Довжина рядка "Помилка при записі" дорівнює 18*

Ініціалізація безрозмірних масивів не тільки полегшує програмування, але й дозволяє змінювати повідомлення, не піклуючись про розмір масивів.

Цей спосіб застосуємо й до багатовимірних масивів. Для цього потрібно вказати всі розміри, крім першого. (Інші розміри потрібні для того, щоб компілятор правильно виконував індексацію масиву.) Це дозволяє створювати таблиці змінної довжини, оскільки компілятор автоматично розміщає їх у пам'яті. Наприклад, оголошення масиву **sqrt** як безрозмірного виглядає так:

int sqrs[ ][2] = {

{1, 1},

{2, 4},

(З, 9},

{4, 16},

{5, 25},

{6, 36},

{7, 49},

{8, 64},

{9, 81},

{10, 100}

};

Перевага такого оголошення складається в тому, що розмір таблиці можна змінювати, не міняючи розміри масиву.

**Використана література**

1. Г. Шилдт, Полный справочник по С++, 4-е видання, в-во «Вильямс», 2006
2. Х.М.Дейтел, Как програмировать на С++, 4-е видання, в-во «Бином-Пресс» 2009.
3. Р. Лафоре, Объектно-ориентированное программирование в С++, в-во «Питер», 2004, с 924.