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,** що має 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 **p[0]** до **p[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],    i;
/*  Вихід індексу масиву  за межі припустимого діапазону  */
for(i=0;  i<100;  i++)  count[i]  =  i;
```

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 a, що починається з адреси 1000 й оголошений за допомогою оператора

```
char a [7];
```

Елемент	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[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   *p;
int  sample[10] ;
p  =   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 **i**.

```
int main(void)
{
   int   i[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 **func1()** доступ до масиву **i**, 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  func1(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","А")** дорівнює 31. Зверніть увагу на те, що значення **stremp** (**"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 <iostream>
using namespace std;
int main()
{
      int  t, i, num[3][4];
      for (t = 0; t < 3; ++t)
            for (i = 0; i < 4; ++i)
                  num[t][i] = (t * 4) + i + 1;
      /*   Вивід  на  екран   */
      for (t = 0; t < 3; ++t) {
            for (i = 0; i < 4; ++i)
                  cout << num[t][i] << "\t";
            cout<<endl;
      }
      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][i]

	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.

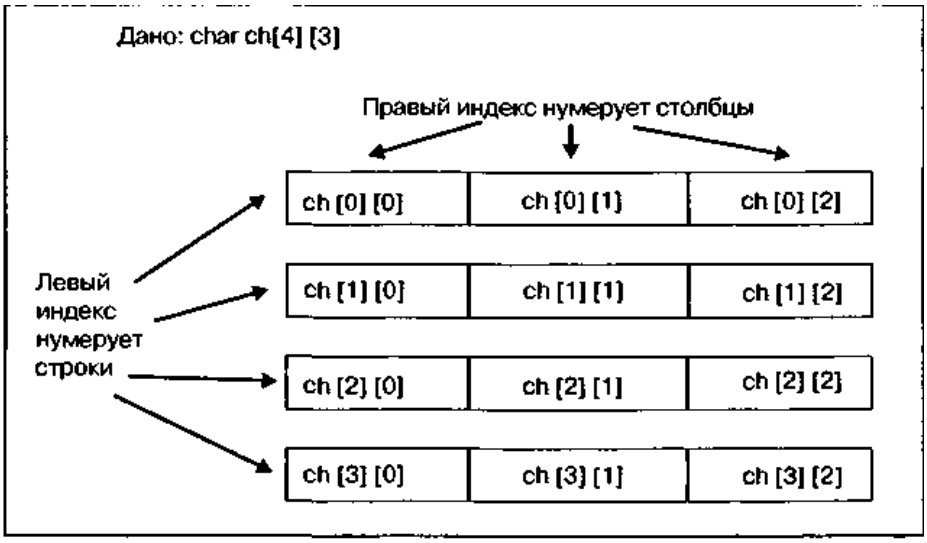


Рис. 4.2. Двухмерный массив

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:

```
x[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 <iostream>
using namespace std;
#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()
{
      char  ch, str[80];
      for (;;) {
            do {
                  cout<<"(В)від оцінок"<<endl;
                  cout<<"(Д)рук оцінок"<<endl;
                  cout<<"(К)інець"<<endl;
                  gets_s(str);
                  ch = toupper(*str); //повертає велику букву з рядкового типу в
символьний
            } while (ch != 'В' && ch != 'Д' && ch != 'К');
            switch (ch) {
            case 'В':
                  enter_grades();
                  break;
            case 'D':
                  disp_grades(grade);
                  break;
            case 'К':
                  exit(0);
            }
      }
      return 0;
}
/* Введення оцінок. */
void enter_grades()
{
      int t, i;
      for (t = 0; t < CLASSES; t++) {
            cout << "Class №: " << t + 1 << endl;
            for (i = 0; i < GRADES; ++i)
                  grade[t][i] = get_grade(i);
            }
}
/* Зчитування оцінки. */
int get_grade(int num)
{
      char s[80];
      cout<<"Введіть оцінку студента № : "<<num + 1<<endl;
      gets_s(s);
      return(atoi(s)); // atoi – переводить число в рядковому форматі і інтовий
}
/* Вивід оцінок. */
void disp_grades(int g[][GRADES])
{
      int t, i;
      for (t = 0; t < CLASSES; ++t) {
            cout<<"Група № : "<< t + 1<<endl;
            for (i = 0; i < GRADES; ++i)
                  cout<<"Студент №"<< i + 1<<" is "<< g[t][i]<<endl;
      }
     }
```

Багатовимірні масиви

У мові С/С++ 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, експоненціально 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(),** що одержує 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

Ініціалізація масиву

У мові С/С++ 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 присвоюється 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 **i** [0] привласнюється значення 1, а елементу **i** [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 групуванням* (subaggregate grouping). Наприклад 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 e2[13] = "Помилка при записі";
char e3[18]  =  "Неможливо відкрити  файл";
```

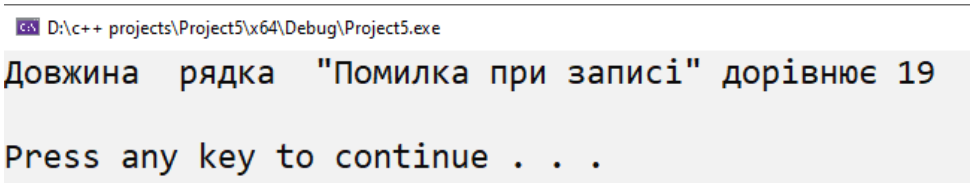
Щоб задати правильний розмір масивів, довелося б підраховувати точну кількість символів у кожному повідомленні. На щастя, компілятор може сам визначити розмір масиву. Якщо в операторі ініціалізації не зазначений розмір масиву, компілятор автоматично створить масив, що вміщає всі початкові значення. Такий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

```
char el[ ] = "Помилка при читанні";
char e2[ ] = "Помилка при записі";
char e3[ ]  =  "Неможливо відкрити файл";
```

В цьому випадку оператор:

```
cout<<"Довжина  рядка  \""<<e2<<"\" дорівнює "<<sizeof(e2)<<endl;
```

виведе на екран повідомлення:



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},
{3,  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.

Цикл foreach

Використання циклу з параметром дає гнучкий механізм реалізації циклічних обчислень та виконання операції, які однакові для значного об’єму даних.

Цикл **for** логічніше використовувати і при застосуванні масивів. Проте при використанні кількох масивів, головно коли аргументи одного масиву беруться з іншого, використання циклу **for** може призвести до технічних помилок, до неврахування певних параметрів.

Тому в С ++ 11 додали новий тип циклу - **foreach** (або «цикл, заснований на діапазоні»), який надає більш простий і безпечний спосіб ітерації по масиву (або будь-який інший структурі даних).

Синтаксис циклу **foreach** наступний:

```
for (оголошення_елемента: масив)
  команди;
```

Виконується ітерація по кожному елементу масиву, привласнюючи значення поточного елемента масиву змінній, оголошеної як елемент (оголошення_елемента). З метою кращої продуктивності голошений елемент повинен бути того ж типу, що й елементи масиву, інакше станеться неявне перетворення типу.

Розглянемо простий приклад з використанням циклу **foreach** для виведення всіх елементів масиву **math**:

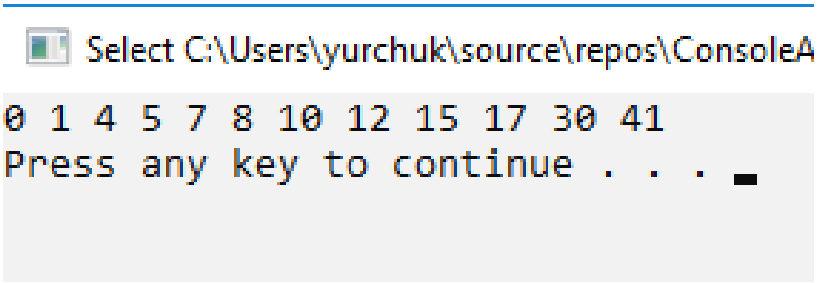
```
#include "windows.h"

#include <iostream>
using namespace std;
int main()
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	int math[] = { 0, 1, 4, 5, 7, 8, 10, 12, 15, 17, 30, 41 };
	for (int number : math)
		cout << number << ' ';

	cout << endl;
	system("pause");
	return 0;

	}
```



Розглянемо докладніше, як це все працює. При виконанні циклу **foreach** змінній **number** присвоюється значення першого елемента (тобто 0). Далі програма виконує набір команд в тілі циклу. В нашому випадку це виведення значення змінної **number**, тобто нуля. Потім цикл **for** виконується знову і значенням **number** вже є 1 (другий елемент масиву). Вивід значення **number** виконується знову. Цикл продовжує виконання до тих пір, поки в масиві не залишиться непройденого елементів. В кінці виконання програма повертає 0 в операційну систему (оператор return).

Зверніть увагу, змінна **number** не є індексом масиву. Їй просто присвоюється значення елемента масиву в поточній ітерації циклу.

Цикл foreach і ключове слово auto

Оскільки оголошений елемент циклу **foreach** повинен бути того ж типу, що й елементи масиву, то це ідеальний випадок для використання ключового слова **auto**, коли ми дозволимо C ++ обчислити тип даних елементів масиву за нас.

Наприклад:

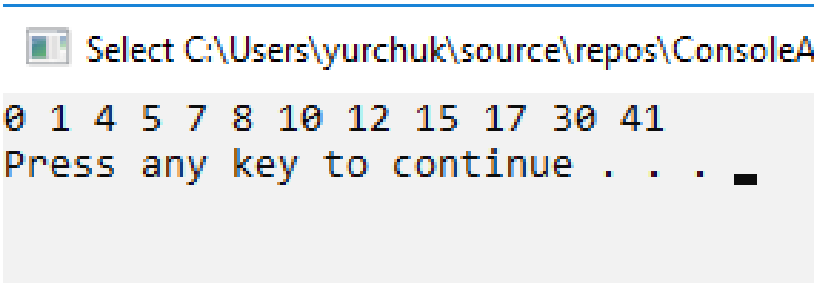
```
#include "windows.h"
```

```
#include <iostream>
using namespace std;
int main()
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	int math[] = { 0, 1, 4, 5, 7, 8, 10, 12, 15, 17, 30, 41 };
	for (auto number : math)
		cout << number << ' ';

	cout << endl;
	system("pause");
	return 0;

	}
```



Цикл foreach і посилання

У прикладах вище оголошений елемент завжди оголошується в якості змінної (значення):

```
int array[7] = { 10, 8, 6, 5, 4, 3, 1 };
for (auto element : array) // element буде копією поточного елемента масиву
cout << element << '';
```

Тобто кожен оброблений елемент масиву копіюється в змінну element. А це копіювання може виявитися витратним, в більшості випадків ми можемо просто посилатися на вихідний елемент за допомогою посилання:

```
int array[7] = { 10, 8, 6, 5, 4, 3, 1 };
for (auto & element : array) // символ амперсанда робить element посиланням на
                             поточний елемент масиву, запобігаючи створення копії
cout << element << '';
```

В наведеному вище прикладі в якості оголошується елемента циклу **foreach** використовується посилання на поточний елемент масиву, при цьому копіювання цього елемента не відбувається. Але із зазначенням звичайного посилання будь-які зміни елемента будуть впливати на сам масив, що не завжди може бути бажано.

Звичайно ж, гарною ідеєю буде зробити оголошений елемент const, тоді ви зможете його використовувати в режимі «тільки для читання»:

```
int array[7] = { 10, 8, 6, 5, 4, 3, 1 };
for (const auto &element : array) // element - це константная посилання на поточний
                                  елемент масиву в ітерації
cout << element << '';
```

Правило: Використовуйте звичайні посилання або константні посилання в якості оголошується елемента в циклі **foreach** (в цілях поліпшення продуктивності).

Використана література

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