

СТРУКТУРНІ (СКЛАДЕНІ) ТИПИ ДАНИХ. МАСИВИ.

ОДНОВИМІРНІ МАСИВИ

Структурні типи даних

Всі типи мови C++ поділяються на категорії:

- фундаментальні (основні) типи, їх ще називають прості (скалярні) типи;
- вказівники;
- структурні типи;
- класи.

Фундаментальні типи в свою чергу поділяються на:

- дискретні : - **int** (цілі числа), **char** (символи), **bool** (логічні значення);
- дійсні типи – **float**, **double** – числа з плаваючою крапкою оддинарної та подвійної точності;
- **void**.

Дискретні типи – це типи, для значень яких існують поняття «наступне» чи «попереднє» значення. Іншими словами, якщо можна вказати, яке значення буде безпосередньо слідувати чи передувати деякому поточному значенню, – то такий тип є дискретним. Дискретні типи в C++ можуть представляти цілі числа, тому їх ще називають цілими типами. В деяких джерелах використовується також термін інтегральні типи.

Типи з плаваючою крапкою дозволяють визначати числа, які можуть мати дробову частину, тому їх ще називають дійсними типами.

Тип **void** описує порожній набір значень. Визначити змінну типу **void** – неможливо. Цей тип використовується в основному для оголошення функцій, які не повертають значення, або універсальних вказівників на не типізовані або довільним чином типізовані дані.

Вказівники – це тип, який можна вважати проміжним між простими і структурними типами. З однієї сторони, його значення – неподільні (як і значення простих типів); з іншої сторони – вказівники визначаються за допомогою інших типів – як структурні типи.

Структурні типи (user defined types – типи, які визначає користувач) – формуються за допомогою поєднання в єдине ціле значень простих типів (і, можливо, вказівників).

До *структурних типів* відносяться:

- регулярні типи

- масиви;
- літерні рядки (це різновид масивів, а саме – одновимірні масиви символів, причому останнім символом мусить бути символ з кодом 0);
- переліки (enum);
- комбіновані типи
- структури (struct);
- об'єднання (union);
- файли;
- динамічні (в т.ч. рекурсивні) структури даних.

Інколи до структурних типів відносять і клас

Можна визначити *будь-який тип* за допомогою ключового слова **typedef**. Якщо перед конструкцією оголошення змінної вказати ключове слово **typedef**, то отримаємо не оголошення змінної, а визначення типу:

```
int a; // оголошення змінної a типу int
typedef int age; // визначення типу A int
age b; // оголошення змінної b типу A
```

Визначивши свій тип, його ім'я можна використовувати скрізь, де допускається використовувати будь-який тип, наприклад, при оголошенні змінних чи параметрів функцій.

Поняття масиву

При використанні простих змінних кожній змінній відводиться окрема область пам'яті, і, відповідно, кожній такій області пам'яті відповідає своє ім'я змінної. Якщо з групою величин одного і того самого типу потрібно виконати деякі однотипні дії, то зручніше надати таким величинам одне ім'я і розрізняти ці величини за їх порядковим номером, – це дозволяє компактно записувати команди виконання зазначених дій за допомогою циклів.

***Масив** – це скінчена іменована область пам'яті, в якій зберігається група однотипних величин.* Скінчена – означає відому та обмежену величину області пам'яті, яка виділяється для масиву. Іменована – означає, що масив має ім'я.

Величини, які входять до складу масиву, називаються його *елементами*. Порядковий номер, за яким розрізняють елементи масиву, називають *індексом*.

Індекси визначають місцезнаходження елемента в масиві.

Елементи нумеруються цілими числами, нумерація індексів завжди *починається з нуля*.

Оскільки всі елементи масиву мають одну і ту саму природу (наприклад, всі –

цілі числа, або всі – дійсні числа, тобто, не допускається випадок, коли частина елементів одного типу, а частина – іншого), то масиви часто називають *регулярними типами*.

Якщо елементи масиву розрізняються за допомогою одного індексу, – тобто, нумерація елементів основана на одному цілому числі, – то такі масиви називають *одновимірними*. Прикладом одновимірного масиву можуть бути пасажирські місця в автобусі – номер місця (одне число) – це індекс елемента, а саме місце – елемент масиву.

Якщо ж елементи масиву розрізняються за допомогою кількох (двох і більше) індексів, – коли нумерація елементів основана на двох і більше цілих числах, – то такі масиви називають *двовимірними* чи, відповідно, *багатовимірними*. Прикладами двовимірних масивів можуть бути пасажирські місця в поїзді (перший індекс – номер вагону, другий індекс – номер місця у вагоні) та місця в кінотеатрі (перший індекс – номер ряду, другий індекс – номер місця у ряді).

Якщо масив містить N елементів, то це означає, що він містить елементи з індексами (порядковими номерами) $0, 1, 2, \dots, N-1$. Елемента з номером N у масиві немає.

Основними відмінностями масива від звичайних змінних є:

- *спільне ім'я для всіх значень;*
- *доступ до певного значення за його індексом;*
- *можливість опрацювання значень у циклі.*

Кожен індекс масива повинен записуватися в окремих квадратних дужках []. Кількість індексів масива необмежена.

Одновимірні масиви

Одновимірний масив – це скінчена іменована область пам'яті, в якій зберігається група однотипних величин, що розрізняються за допомогою однієї характеристики – індексу.

Одновимірний масив *оголошується* так:

`<тип_даних> <ім'я_масива> [<розмір>];`

де `тип_даних` – тип даних елементів масива, при цьому елементами масива не можуть бути функції та елементи типу `void`; `розмір` – кількість елементів масива.

Наприклад, команда: `int arr[6];` – оголошує одновимірний масив (так званий «вектор») із 6 елементів цілого типу. Елементи масиву нумеруються, починаючи від нуля, тобто масив `arr` складається із 6-ти елементів цілого типу, індекс (номер) яких мусить мати значення від 0 до 5.

Кількість елементів масиву разом з типом його елементів визначає об'єм пам'яті, необхідний для розміщення масиву, – цей об'єм визначається на етапі компіляції, і тому кількість елементів масиву може бути вказана лише цілою додатною константою або константним виразом.

Для доступу до елементів одновимірного масива використовують одну з таких форм:

- індексну: `<ім'я_масива> [<індекс>]`
- адресну (вказівникову): `*(<ім'я_масива> + <індекс>)`

Переважно використовується індексна форма доступу, оскільки вона більш зручна для людського сприйняття.

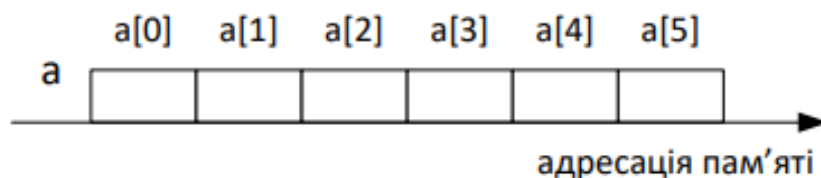
Оскільки в мовах C/C++ нумерація індексів починається з нуля, то значення індексів мають знаходитися в діапазоні від нуля до величини, на одиницю меншу ніж розмір масива.

Якщо вказати недопустиме значення індексу (для попереднього прикладу: менше 0 або більше 5), то компілятор не виявить цієї помилки і при виконанні програми відбудеться звертання до області пам'яті, яка не відведена для масиву. Це може привести до непередбачуваних наслідків! При звертанні до елементів масиву автоматичний контроль виходу індексу за межі масиву не здійснюється, – щоб уникнути помилок, це має робити програміст

Розглянемо оголошення деяких одновимірних масивів:

```
int arr[12];    // масив 12-ти цілих чисел
double vector[15]; // масив 15-ти дійсних чисел
char test[20]; // масив 20-ти символів
```

В пам'яті одновимірний масив займає неперервну область та зберігається так, що в комірках пам'яті з молодшими адресами зберігаються елементи масиву з меншими значеннями індексів



Масиви, як і змінні, можна ініціалізовувати при оголошенні. Якщо реальна кількість ініціалізованих значень є меншою за розмір числового масива, то всі неініціалізовані елементи автоматично отримують значення 0.

Початкові значення для масивів записуються всередині фігурних дужок: константи, які стануть значеннями елементів масиву записуються через кому у фігурних дужках `{}`.

Одновимірний масив розглядається як вектор, його ініціалізація можлива такими способами:

1) Кількість елементів в оголошенні масиву точно збігається з кількістю констант, вказаних у фігурних дужках:

```
int a[6] = { 9, 8, 7, 6, 5, 4 };
```

Значення елементам присвоюються по черзі: перша константа в ініціалізаторі – першому елементу масиву (із значенням індексу, що дорівнює 0), друга константа в ініціалізаторі – другому елементу масиву (із значенням індексу, що дорівнює 0) і т.д. Попередня команда еквівалентна наступним:

```
int a[6];
```

```
a[0] = 9;
a[1] = 8;
a[2] = 7;
a[3] = 6;
a[4] = 5;
a[5] = 4;
```

a	9	8	7	6	5	4
	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

2) Кількість елементів в оголошенні масиву більша кількості констант, вказаних у фігурних дужках:

```
int a[6] = { 9 };
```

Тоді елементи, значення яких не вказане, отримують нульові значення. Для цього потрібний ініціалізатор, бо оголошений, але не ініціалізований локальний масив (як і проста локальна змінна) значень за умовчанням не отримує. Попередня команда еквівалентна наступним:

```
int a[6];
```

```
a[0] = 9;
a[1] = 0;
a[2] = 0;
a[3] = 0;
a[4] = 0;
a[5] = 0;
```

a	9	0	0	0	0	0
	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

3) Кількість елементів в оголошенні масиву можна не вказувати, але тоді має бути вказаний ініціалізатор – кількість елементів обчислюється на основі кількості констант, вказаних у фігурних дужках:

```
int a[] = { 9, 8, 7, 6, 5, 4 };
```

Ця команда еквівалентна наступним

```
int a[6];
```

```
a[0] = 9;
a[1] = 8;
a[2] = 7;
a[3] = 6;
a[4] = 5;
a[5] = 4;
```

a	9	8	7	6	5	4
	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

Також для оголошення масивів можна використовувати *типізовані константи-масиви*, які дають змогу водночас оголосити масив і задати його значення як константи:

```
const <тип_даних> <ім'я_масива> [<розмір_масива>] =
    {<значення_елементів_масива>};
```

Змінювати значення елементів констант-масивів не можна.

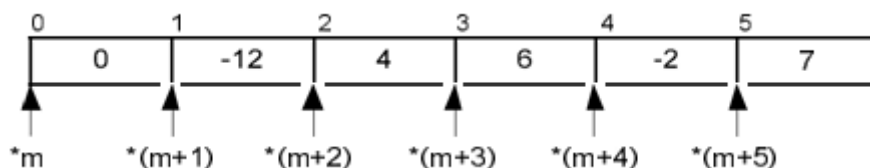
Розглянемо оголошення деяких одновимірних констант-масивів:

```
const int arr[5] = { 9, 3, 7, 0, 13 };
const int c[5] = { 15, 6, 54 }; //два елементи ініціалізовані нулями:
const float f[5] = { 1.5, 6, 8, 7.4, 1.15 }; // масив 5 дійсних чисел
```

Масиви і вказівники в мовах C/C++ можуть використовуватись взаємозамінно. Під час програмної реалізації індексний спосіб доступу до елемента масива зводиться до вказівникового, оскільки операції над вказівниками виконуються швидше. Нехай маємо вказівник на одновимірний масив цілих чисел:

```
int m[6] = { 0, -12, 4, 6, 2, -7 };
```

При такому оголошенні масива пам'ять виділяється не лише для шести елементів масива, а й для вказівника з ім'ям *m*, значення якого є адресою першого елемента масива (*m[0]*), тобто сам масив залишається безіменним, а доступ до елементів масива здійснюється через вказівник з ім'ям *m*.



Якщо елементи масива опрацьовуються по чергову, то доцільніше використовувати вказівниковий спосіб доступу. Якщо ж вибір елементів є випадковим, то краще використовувати індексний спосіб, який є більш наочним та зручним для сприйняття. Оскільки ім'я масива є вказівником на його перший елемент, то іншому вказівнику можна присвоїти адресу першого елемента масива за допомогою операції присвоєння:

```
int m[6] = { 0, -12, 4, 6, -2, 7 };
// оголошення та ініціалізація вказівника адресою початку масива
```

```
int* w = m;    // ця операція еквівалентна операції *w = &m[0];
// оголошення вказівника
int *ptr;
// ініціалізація вказівника адресою початку масива
ptr = m; // ця операція еквівалентна операції ptr = &m[0];
```

Тут обидва вказівники вказують на один і той же масив. Індексація вказівників виконується аналогічно до індексації масивів. Наприклад, щоб присвоїти першому елементу масива значення 2, можна скористатись одним із таких операторів:

```
* m = 2; // *(m + 0) = 2;
*ptr = 2; // *(ptr + 0) = 2;
m[0] = 2;
ptr[0] = 2;
```

Дії над масивами

Всі дії над масивами виконуються по-елементно. С / С++ не дозволяє присвоєння масивів один одному. Розглянемо приклад: обчислення суми елементів масиву

```
#include <iostream>
using namespace std;

int main()
{
    const int n = 10;
    int a[n] = { 9, 8, 7, 6, 5 };

    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += a[i];
    cout << "sum = " << sum << endl;
    return 0;
}
або
#include <iostream>
using namespace std;

int sumArr(int[]);

int main()
{
    const int n = 10;
    int a[n] = { 9, 8, 7, 6, 1 };

    int sum = 0;
    for (int i = 0; i < n; i++)
```

```

        sum += a[i];
    cout << "sum = " << sumArr(a) << endl;
    return 0;
}
int sumArr(int a[])
{
    int sum = 0;
    for (int i = 0; i < 10; i++)
        sum += a[i];
    return sum;
}

```

Кількість елементів масиву доцільно вказувати за допомогою іменованої константи, – як в наведеному прикладі, – тоді програму значно легше модифікувати

Стилістично правильне введення елементів одновимірного масиву розглянемо на прикладі сортування масиву цілих чисел методом вибору.

Переглядаємо всі елементи масиву (починаючи з першого, – з індексом, рівним 0). Вибираємо мінімальний елемент масиву і міняємо місцями його з першим елементом. Потім переглядаємо елементи, починаючи з другого (індекс якого рівний 1), вибираємо мінімальний з них і міняємо місцями з другим елементом. Далі переглядаємо елементи, починаючи з третього (індекс якого рівний 2), вибираємо мінімальний з них і міняємо місцями з третім елементом. Вказані дії повторюємо $n-1$ раз, щоразу збільшуючи номер позиції елемента, з якого починається пошук мінімального.

```

#include <iostream>
using namespace std;
int main()
{
    const int n = 10; // кількість елементів масиву
    int a[n]; // оголосили масив

    // введення елементів масиву
    for (int i = 0; i < n; i++) // в циклі для всіх елементів масиву
    {
        cout << "a[" << i << "]= ? "; // вивели підказку
        cin >> a[i]; // ввели елемент масиву
    }

    // впорядкування масиву
    for (int i = 0; i < n - 1; i++)
    {
        // n-1 раз шукаємо мінімальний елемент
        int imin = i; // вважаємо i-й елемент мінімальним
        for (int j = i + 1; j < n; j++)
            if (a[j] < a[imin]) // якщо поточний елемент менший

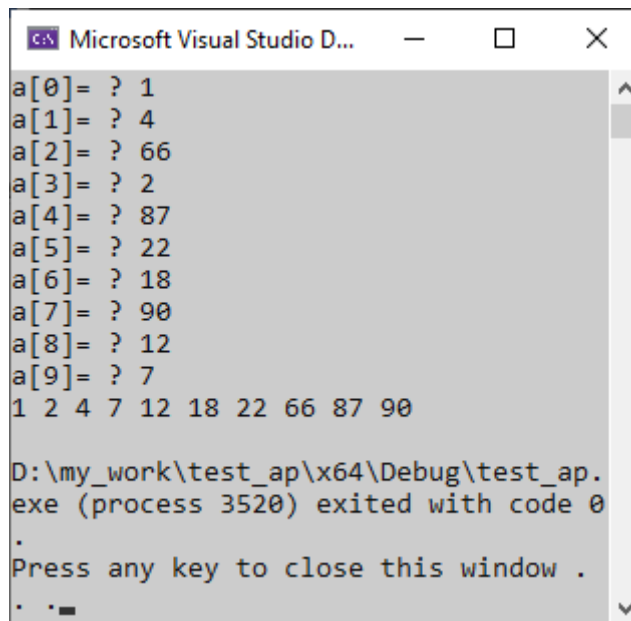
```



```

        imin = j; // того, який вважаємо мінімальним
    // - запам'ятали його індекс
    // обмін місцями елементів з номерами i та imin
    int tmp = a[i];
    a[i] = a[imin];
    a[imin] = tmp;
}
// виведення впорядкованого масиву
for (int i = 0; i < n; i++)
    cout << a[i] << " ";
cout << endl;
return 0;
}

```



```

Microsoft Visual Studio D...
a[0]= ? 1
a[1]= ? 4
a[2]= ? 66
a[3]= ? 2
a[4]= ? 87
a[5]= ? 22
a[6]= ? 18
a[7]= ? 90
a[8]= ? 12
a[9]= ? 7
1 2 4 7 12 18 22 66 87 90

D:\my_work\test_ap\x64\Debug\test_ap.
exe (process 3520) exited with code 0
.
Press any key to close this window .
.

```

Також можна отримати значення масива за допомогою генератора випадкових чисел:

```

#include <iostream>
using namespace std;
int main()
{
    const int n = 10; // кількість елементів масиву
    int a[n]; // оголосили масив

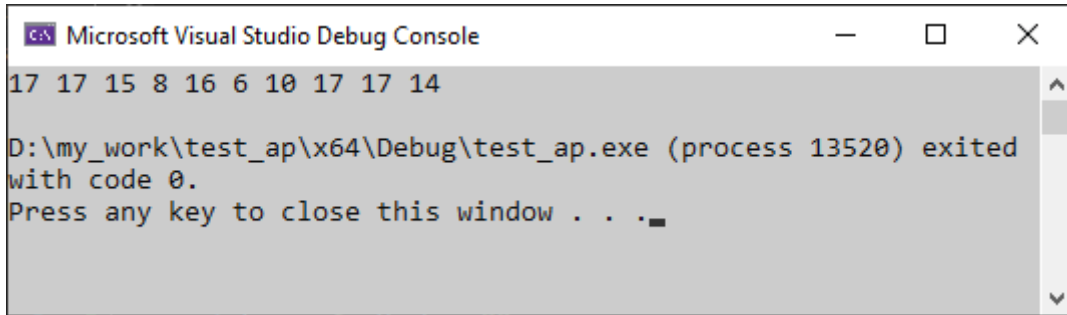
    int Low = 6, High = 20;
    srand((unsigned)time(NULL));
    // заповнення масиву випадковими значеннями
    for (int i = 0; i < n; i++)
    {
        a[i] = Low + rand() % (High-Low + 1);
    }
    // виведення масиву
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    cout << endl;
}

```

```

return 0;
}

```



```

Microsoft Visual Studio Debug Console
17 17 15 8 16 6 10 17 17 14
D:\my_work\test_ap\x64\Debug\test_ap.exe (process 13520) exited
with code 0.
Press any key to close this window . . .

```

Виведення елементів масива у стовпчик:

```

for (int i = 0; i < n; i++)
    cout << "a[" << i << "]= " << a[i] << endl;

```

Виведення елементів масива, розділених символом табуляції, у рядок:

```

for (int i = 0; i < n; i++)
    cout << "a[" << i << "]= " << a[i] << "\t";

```

Виведення елементів масива за допомогою встановлення ширини:

```

for (int i = 0; i < n; i++)
    cout << setw(5) << "a[" << i << "]= " << setw(2) << a[i];

```

Передавання одновимірних масивів у функції

При передаванні масиву у функцію передається адреса його першого елемента. Таким чином, масив завжди передається за адресою, а не за значенням. При цьому інформація про кількість елементів масиву – втрачається, і тому кількість елементів слід передавати за допомогою окремого параметру.

Якщо кількість елементів масиву – константа, то її можна оголосити глобальною і використовувати у всіх функціях; хоча це – поганий стиль. Значно краще інформацію про кількість елементів передавати не за допомогою глобальних величин, а у вигляді параметрів.

Приклад: ітераційна функція, яка повертає добуток елементів масиву цілих чисел.

```

#include <iostream>
using namespace std;

int MultArr(int mas[], const int n);

int main()
{
    const int k = 5; // кількість елементів масиву
    int a[k] = {3,4,5,2,1 }; // оголосили та ініціалізували масив
    cout << "the product of array elements = " << MultArr(a, k) <<
endl;
    return 0;
}

```

```

}

int MultArr(int mas[], const int n) // функція для обчислення добутку
{
    int mult = 1;
    for (int i = 0; i < n; i++)
        mult *= mas[i];
    return mult;
}
    
```

Можуть бути інші варіанти заголовку (і прототипу) функції:

```

int MultArr(int* mas, const int n); // mas – вказівник на масив
int MultArr(int mas[n], const int n); //n в описі параметра mas ігнорується
    
```

Двовимірні масиви

Оголошення двовимірного масива має такий вигляд:

```
<тип_даних> <ім'я_масива> [<к-ть_рядків>] [<к-ть_стовпчиків>];
```

Кількість елементів масива дорівнює добутку кількості елементів масива за кожним індексом. Наприклад, двовимірний масив з 3-х рядків та 4-х стовпчиків (12-ти елементів) цілого типу:

```
int a[3][4];
```

Структура цього масива така:

```

a[0][0]  a[0][1]  a[0][2]  a[0][3]
a[1][0]  a[1][1]  a[1][2]  a[1][3]
a[2][0]  a[2][1]  a[2][2]  a[2][3]
    
```

Таким чином під масив автоматично виділяється пам'ять, необхідна для розміщення всіх його елементів. При оголошенні масива його можна ініціалізовувати початковими значеннями. *Якщо значень менше ніж елементів, то всі неініціалізовані елементи числових масивів автоматично отримують значення 0.*

```
int w[3][3] = { {2, 3, 4}, {3, 4, 8}, {1, 0, 9} };
```

Для доступу до елементів двовимірного масива використовують одну з таких форм доступу:

індексну:

```
<ім'я_масива>[<індекс_рядка>][<індекс_стовпчика>]
```

адресну (вказівникову) – переважно використовується у функціях:

```

*(<ім'я_масива> + <індекс_рядка> * <кількість_стовпчиків> +
    <індекс_стовпчика>)
    
```

Розглянемо детальніше. У масиві `a[3][5]`, наведеному на рисунку нижче, перші п'ять елементів належать 0-му рядку, наступні п'ять – 1-му, а останні п'ять – 2-му. Оскільки `a` – це вказівник на початок масива, тобто на елемент `a[0][0]`, то

для того, щоб звернутись до елемента $a[1][3]$, потрібно “перестрибнути” від початку масива через 5 елементів нульового рядка і 3 елементи першого рядка:
 $\ast (a+1\ast 5+3)$.



Ініціалізація двовимірного масиву можлива двома способами

1) матриця розглядається як *вектор векторів*, і значення елементів кожного вектора записуємо у фігурних дужках $\{\}$; при цьому кількість елементів для першої (лівої) розмірності можна не вказувати:

```
int m[][2] = { {1,1}, {0,2}, {1,0} };
```

або

```
int m[3][2] = { {1,1}, {0,2}, {1,0} };
```

2) матрицю трактуємо як *одновимірний масив* (утворений із рядків матриці, записаних послідовно один за одним); вказуємо загальний перелік елементів у тому порядку, в якому вони розміщені в пам'яті:

```
int m[3][2] = { 1,1,0,2,1,0 };
```

Введення двовимірних масивів

Введення-виведення значень елементів масива можна виконувати лише поелементно. Для цього потрібно організувати цикли, в яких значення індексів елементів будуть змінюватися послідовно.

- *Введення елементів масива з клавіатури:*

```
const int n = 3, m = 4;
int matrix[n][m];

for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
    {
        cout << "a[" << i << "][" << j << "]="; cin >> matrix[i][j];
    }
```

- *Введення масива за допомогою генератора випадкових чисел:*

```
const int n = 3, m = 4;
int matrix[n][m];

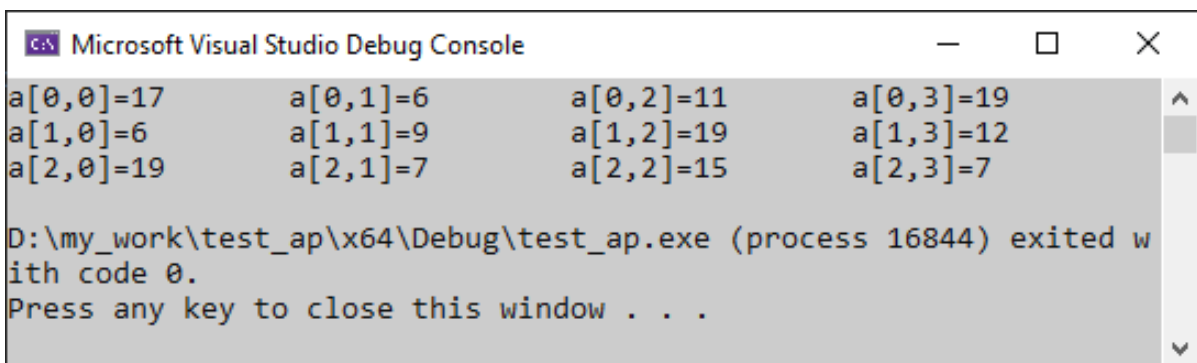
srand((unsigned)time(NULL));
int Low = 6, High = 20;
for (int i = 0; i < n; i++)
```

```
for (int j = 0; j < m; j++)
    matrix[i][j] = Low + rand() % (High - Low + 1);
```

Виведення двовимірних масивів

- Виведення елементів масива, розділених символом табуляції, по рядках

```
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
    {
        cout << "a[" << i << ", " << j << "]= " << matrix[i][j] << "\t";
    }
    cout << endl;
}
```



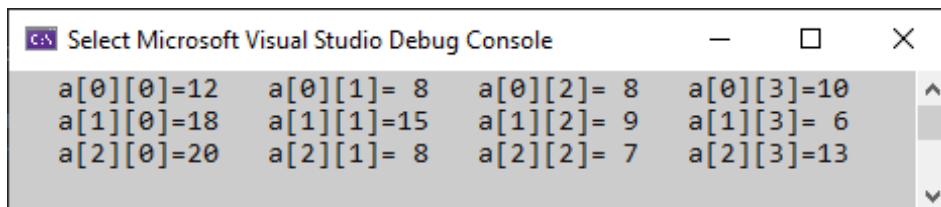
```
Microsoft Visual Studio Debug Console

a[0,0]=17      a[0,1]=6      a[0,2]=11      a[0,3]=19
a[1,0]=6       a[1,1]=9       a[1,2]=19      a[1,3]=12
a[2,0]=19      a[2,1]=7       a[2,2]=15      a[2,3]=7

D:\my_work\test_ap\x64\Debug\test_ap.exe (process 16844) exited with code 0.
Press any key to close this window . . .
```

- Виведення елементів масива по рядках за допомогою встановлення ширини

```
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++)
    {
        cout << setw(5) << "a[" << i << "][" << j << "]= " << setw(2) << matrix[i][j];
    }
    cout << endl;
}
```



```
Select Microsoft Visual Studio Debug Console

a[0][0]=12    a[0][1]= 8    a[0][2]= 8    a[0][3]=10
a[1][0]=18    a[1][1]=15    a[1][2]= 9    a[1][3]= 6
a[2][0]=20    a[2][1]= 8    a[2][2]= 7    a[2][3]=13
```

Якщо масив передається у функцію, то передається адреса його першого елемента. Тому масив завжди передається за адресою, а не за значенням. При цьому інформація про кількість елементів масиву – втрачається, і тому кількість елементів за кожним виміром потрібно передавати за допомогою окремих параметрів:

```
int sum(const int* a, const int rowCount, const int colCount) { ... }
#include <iostream>
#include <iomanip>
```

```
using namespace std;

int sumArr(const int arr[4][5]);

void main() {

    // розміри масива мають бути константами
    const int n = 4, m = 5;
    int b[n][m];
    //налаштуємо генерацію випадкових чисел
    const int Low = 6, High = 20;
    srand((unsigned)time(NULL));
    // ініціалізація і виведення масива
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            b[i][j] = Low + rand() % (High - Low + 1); cout << setw(7) << b[i][j];
        }
        cout << endl;
    }
    // виведення суми
    cout << "Sum = " << sumArr(b);

}

int sumArr(const int arr[4][5]) {
    int s = 0;
    for (int i = 0; i < 4; i++) for (int j = 0; j < 5; j++)
        s += arr[i][j];
    return s;
}
```

```
Microsoft Visual Studio Debug Console

18    15    13    10     6
11     9    18    17    12
16    13    15     8    19
 9     8    20    19     6

Sum = 262
D:\my_work\test_ap\x64\Debug\test_ap.exe (process 13104) exited with code 0.
Press any key to close this window . . .
```