



СТРУКТУРНІ (СКЛАДЕНІ) ТИПИ ДАНИХ. МАСИВИ.

ОДНОВИМІРНІ МАСИВИ

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.

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

Тип `void` описує порожній набір значень.

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.

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

Поняття масиву

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.

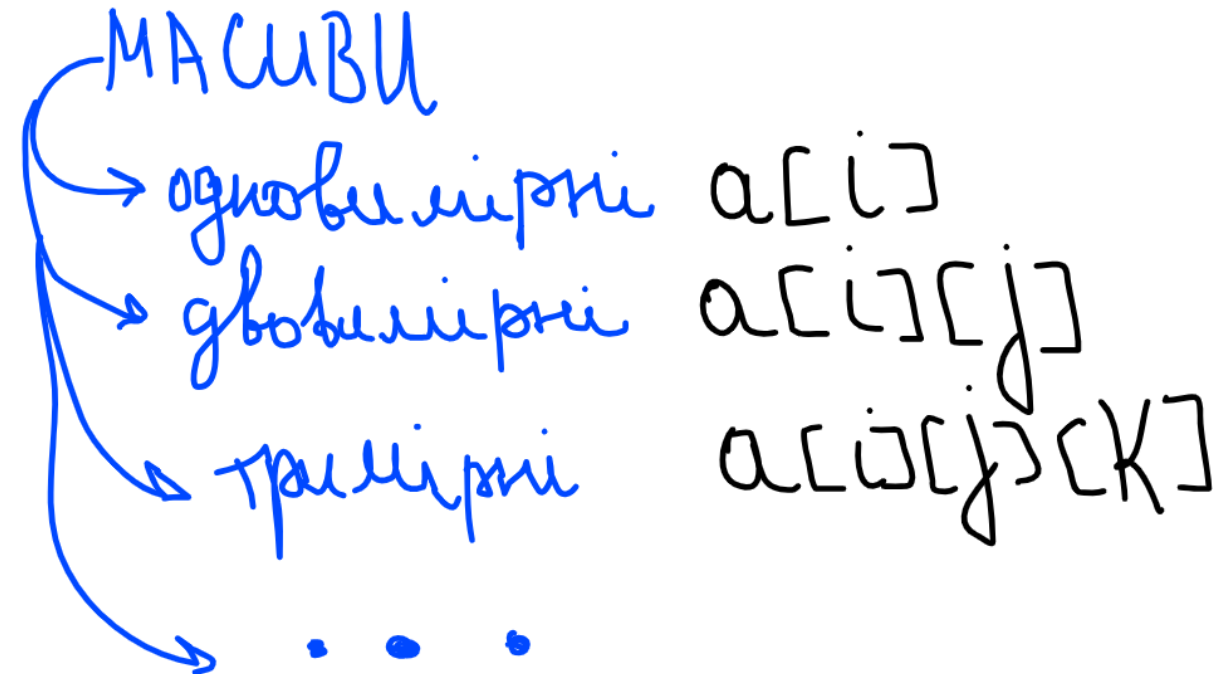
Величини, які входять до складу масиву, називаються його *елементами*.

Порядковий номер, за яким розрізняють елементи масиву, називають *індексом*.

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

Основними відмінностями масива є:

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



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.

```
int arr[6];
```

Елементи масиву нумеруються, починаючи від нуля.
масив `arr` складається із 6-ти елементів цілого типу, індекс (номер) яких мусить мати значення від 0 до 5.

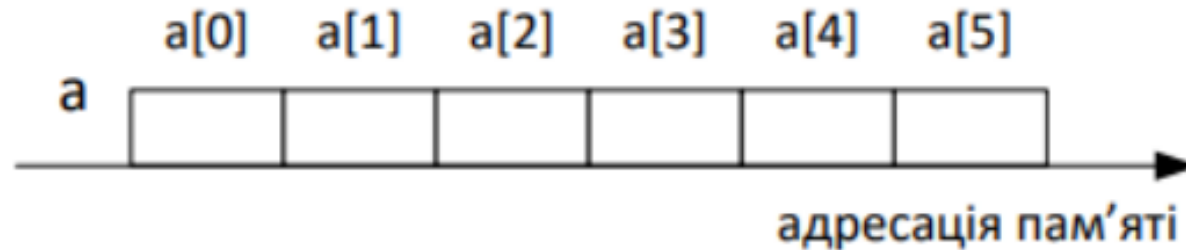
Для доступу до елементів одновимірного масива використовують

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

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



При звертанні до елементів масиву автоматичний контроль виходу індексу за межі масиву не здійснюється ➡ це має робити програміст.

ініціалізація

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

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

```
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 };
```

```
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]

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

Змінювати значення елементів констант-масивів не можна.

```
const int arr[5] = { 9, 3, 7, 0, 13 };
```

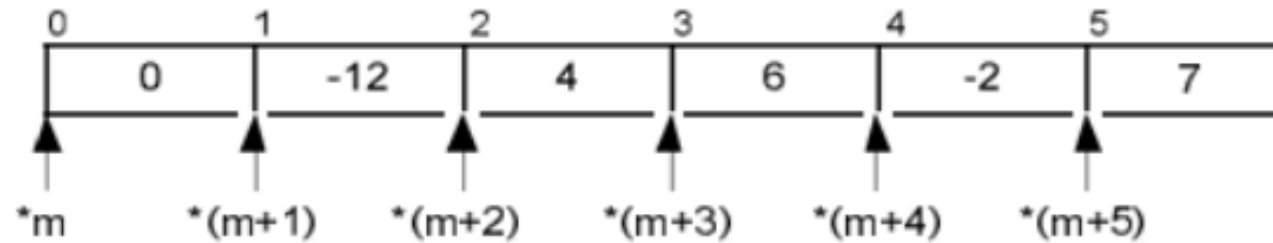
```
const int c[5] = { 15, 6, 54 }; //два елементи ініціалізовані нулями:
```

```
const float f[5] = { 1.5, 6, 8, 7.4, 1.15 };// масив 5 дійсних чисел
```



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

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*.



```
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];
```

```
* 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. !

```
#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;
}
```

```
#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;
}
```





```
#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
```

```
#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
```

Передавання одновимірних масивів у функції

При передаванні масиву у функцію передається адреса його першого елемента.

Якщо кількість елементів масиву – константа, то її можна оголосити глобальною і використовувати у всіх функціях; хоча це – поганий стиль.

```
#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;
}
```

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];
```


Порвернення одновимірних масивів з функції

Функція може повертати вказівник на масив чисел. Функція, що повертає вказівник на одновимірний масив, оголошується таким чином:

```
<тип_даних>* <ім'я_функції> (<тип_даних> <ім'я_масива>  
[, <тип_даних2> <параметр2>], ...)  
  
{<тіло_функції>}
```

// Функція для введення елементів вручну

```
int* inputArray(int arr[], int countelem)  
{  
    cout << "Введіть елементи масиву:\n";  
    for (int i = 0; i < countelem; i++) {  
        cout << "arr[" << i << "]=";  
        cin >> arr[i];  
    }  
    return arr;  
}
```

// Функція для генерації елементів випадково

```
int* generateRandomArray(int arr[], int countelem, int min=1, int max=20)  
{  
    srand((unsigned)time(NULL));  
  
    for (int i = 0; i < countelem; i++)  
    {  
        arr[i] = min + rand() % (max - min + 1);  
    }  
    return arr;  
}
```

Викликати ці функції можна наступним чином

```
const int countelem = 5;  
int arr[countelem], arr1[countelem];  
//генеруємо вручну  
int* f = inputArray(arr, countelem);  
//рандомно  
int* s = generateRandomArray(arr1, countelem)  
//або вказуючи явно діапазон  
int* s = generateRandomArray(arr1, countelem, 1, 50);
```

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]

```
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:

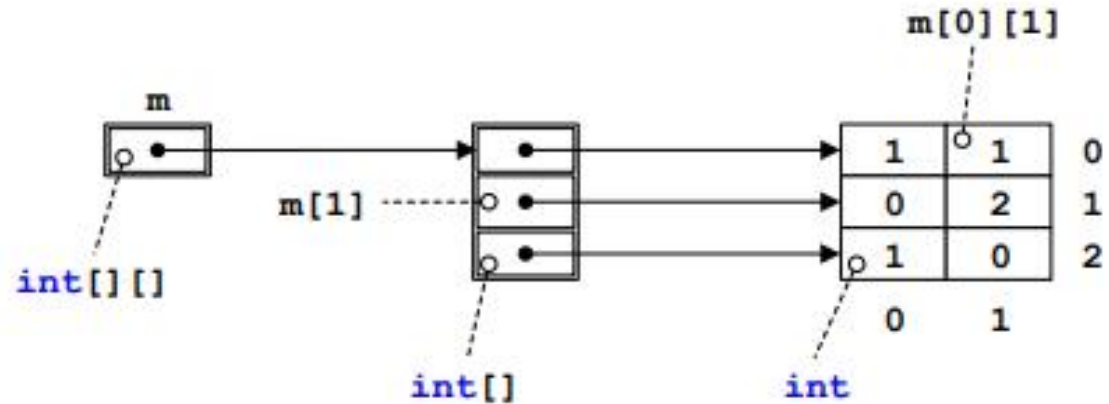
```
<ім'я_масива> [<індекс_рядка>] [<індекс_стовпчика>]
```

адресну (вказівникову) – переважно використовується у функціях:

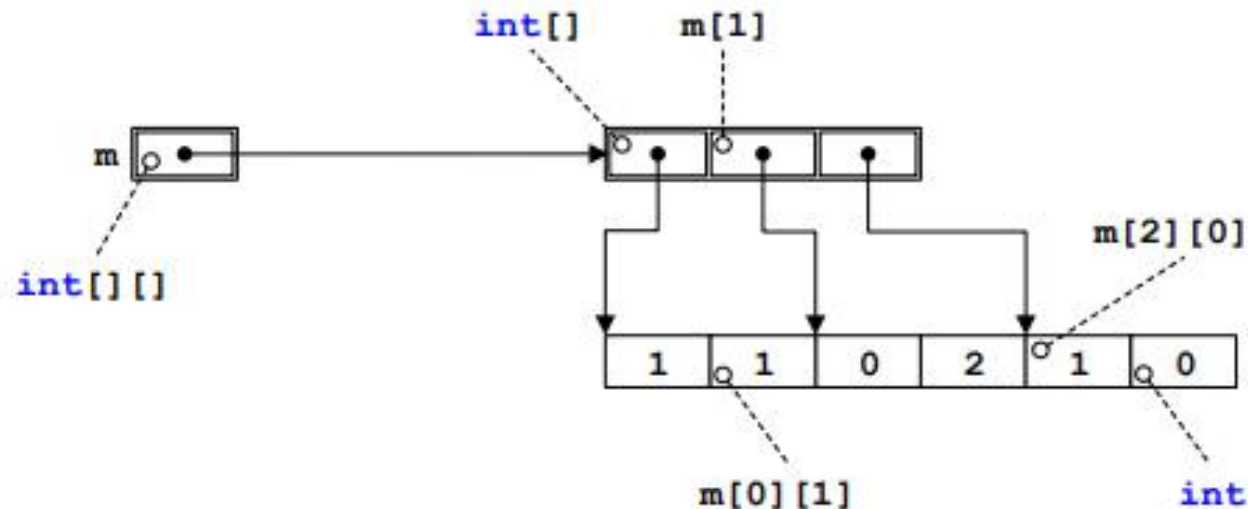
```
*(<ім'я_масива> + <індекс_рядка> * <кількість_стовпчиків> +  
  <індекс_стовпчика>)
```



Звичайні масиви реалізовані в мові C/C++ як константні вказівники, тому m – це константний вказівник на вектор-стовпчик із трьох елементів, кожний з яких, своєю чергою, є константним вказівником на вектор-рядок із двох елементів цілого типу:



Звісно, в оперативній пам'яті немає ні рядків, ні стовпчиків. Наведений рисунок – наочний, проте не зовсім точний. Правильніше буде нарисувати розподіл пам'яті лінійно –



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 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);
```

Введення двовимірних масивів

- 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++)
```

```
    {
```

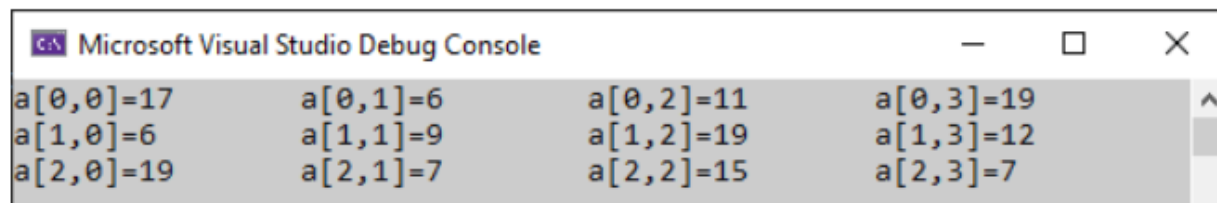
```
        cout << "a[" << i << "][" << j << "]="; cin >> matrix[i][j];
```

```
    }
```


Виведення двовимірних масивів

- Виведення елементів масива, розділених символом табуляції, по рядках

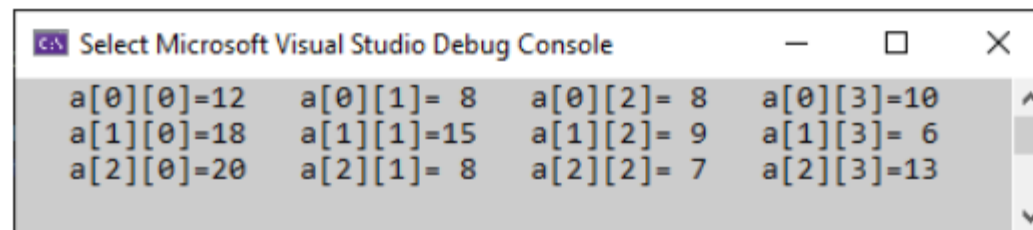
```
for (int i = 0; i < n; i++)  
{  
    for (int j = 0; j < m; j++)  
    {  
        cout << "a[" << i << ", " << j << "]= " << matrix[i][j] << "\t";  
    }  
    cout << endl;  
}
```



```
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
```

- 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
```

```
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];
    //налаштовуємо 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 . . .

ДИНАМІЧНА ПАМ'ЯТЬ. ДИНАМІЧНІ МАСИВИ.

Динамічна пам'ять (heap, купа) – це пам'ять, в якій змінні розміщуються динамічно. Основна потреба у динамічному виділенні пам'яті виникає тоді, коли розмір або кількість даних, які необхідно виділити чи використати, наперед невідомі і визначаються у процесі виконання програми.

Час життя динамічного об'єкта – це час від моменту його створення до знищення, тобто динамічно виділена пам'ять існує поки її не звільнити.

У мові C++ для виділення динамічної пам'яті використовується оператор

new, який має такий синтаксис:

```
<тип_даних> *<змінна> = new <тип_даних> (<змінна>);
```

Динамічна пам'ять, виділена за допомогою оператора **new**, автоматично не звільняється, тому її обов'язково *потрібно звільняти самотійно*

```
delete [] <вказівник>
```

```
int* n = new int;      // виділення пам'яті
*n = 10; // ініціалізація динамічної змінної
delete n; // звільнення пам'яті
```

Або так

```
int n = 10;
// виділення пам'яті та ініціалізація
int *q = new int(n);
delete q; // звільнення пам'яті
```

Динамічні одновимірні масиви

Робота з динамічними масивами відбувається за допомогою змінних типу вказівник. Після створення масива така змінна містить адресу його першого елемента.

Синтаксис оголошення динамічного одновимірного масива за допомогою оператора `new` такий:

```
<тип_даних> *<ім'я_вказівника> = new <тип_даних>
                               [кількість_елементів];
```

Оператор `new` виділяє місце в пам'яті для певної кількості елементів певного типу, а адреса цієї ділянки пам'яті записується у змінну-вказівник.

Для доступу до значень елементів

- індексна форма:

```
<ім'я_масива>[<індекс>]
```

- вказівникова форма:

```
*(<ім'я_масива> + <індекс>)
```

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

int main() {
    // кількість елементів масива
    int n;
    cout << "Enter number of elements: ";
    cin >> n; // зчитати кількість елементів масива
    // створення динамічного одновимірного масива
    int *b = new int [n];
    // перевірка, чи масив створився
    if (b == nullptr) {
        cout << "\nMemory error!\n";
        return -1; // якщо масив не створився – вихід
    }
    // введення масива
    cout << "\n\nEnter elements:\n";
    for (int i = 0; i < n; i++) {
        cout << "b[" << i+1 << "]=";
        cin >> b[i];
    }

    // опрацювання масива
    // звільнення пам'яті, виділеної під масив
    delete [] b;

    return 0;
}
```


Динамічні двовимірні масиви

Динамічний двовимірний масив як одновимірний

```
<тип_даних> *<ім'я_вказівника> = new <тип_даних>
    [кількість_елементів];
```

виділення пам'яті під масив

```
int *m = new int [rowCount * colCount];
```

звільнення пам'яті

```
delete [] m;
```

індексний спосіб:

```
<ім'я_масива> [ <індекс_рядка> * <кількість_стовпчиків> +
    <індекс_стовпчика> ]
```

адресний спосіб:

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

Динамічний двовимірний масив як двовимірний

*спочатку виділяється пам'ять для стовпчика
вказівників на рядки матриці, а потім окремо на кожний рядок.*

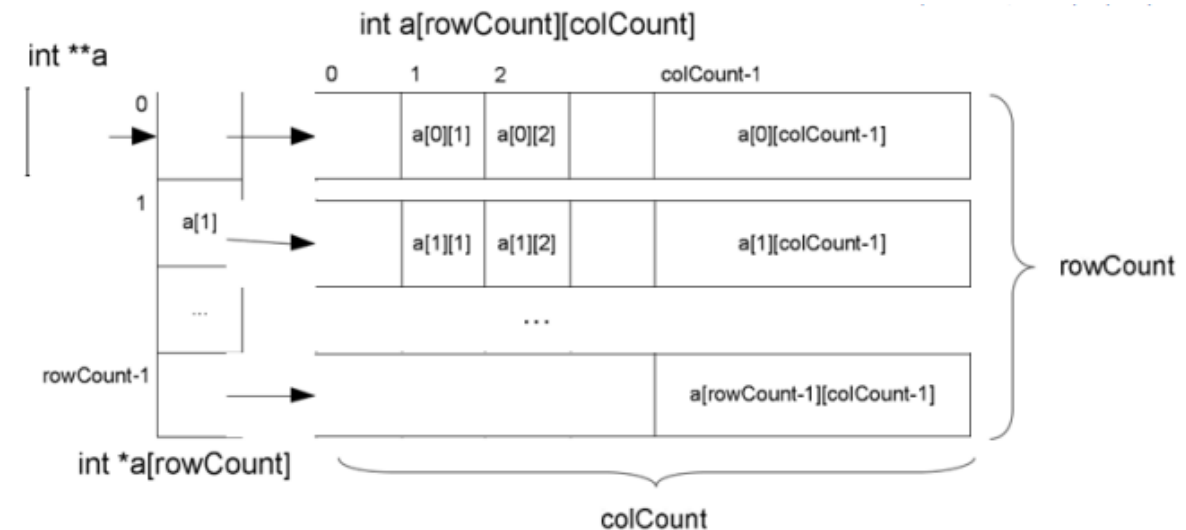
Адреси нульових елементів динамічного двовимірного масива зберігаються в допоміжному масиві, пам'ять під який потрібно виділяти завчасно

(індексна форма):

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

(вказівникова форма):

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



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

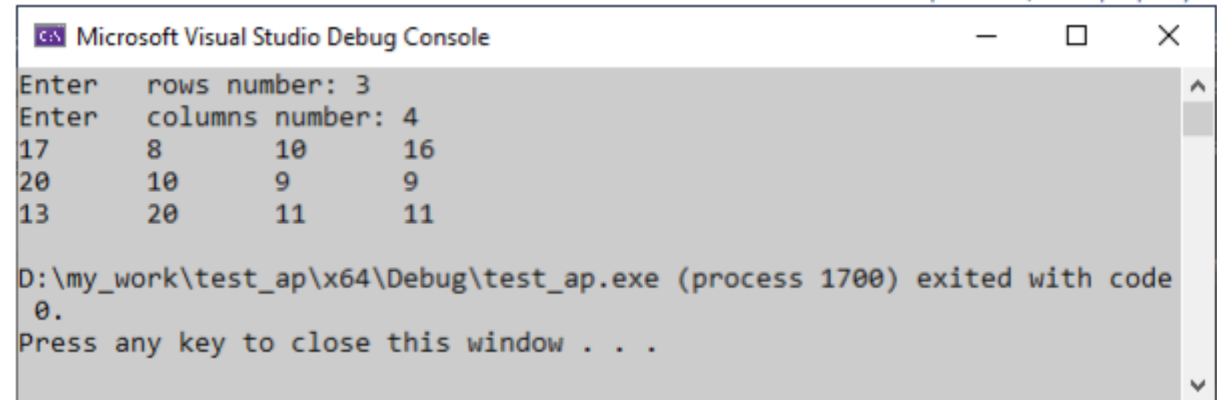
void main() {
    const int Low = 6, High = 20;
    int rowN, // кількість рядків
        colN; // кількість стовпчиків
    cout << "Enter rows number: ";
    cin >> rowN;
    cout << "Enter columns number: ";
    cin >> colN;
    // оголошення та розміщення в пам'яті допоміжного масива
    int** m = new int* [rowN];
    // оголошення та розміщення в пам'яті основного масива
    for (int i = 0; i < rowN; i++)
        m[i] = new int[colN];

    // ініціалізація елементів та виведення масива на екран
    for(int i = 0; i < rowN; i++) {
        for (int j = 0; j < colN; j++) {
            // ініціалізація
            m[i][j] = Low + rand() % (High - Low + 1);

            // індексний спосіб
            cout << m[i][j]<<"\t" ;    // виведення
            // вказівниковий спосіб
            //cout << setw(5) << (*(m + i) + j);
        }
        cout << endl;
    }

    // звільнення пам'яті, виділеної під основний масив
    for (int i = 0; i < rowN; i++)
        delete[] m[i];

    // звільнення пам'яті, виділеної під допоміжний масив
    delete[] m;
}
```



Microsoft Visual Studio Debug Console

```
Enter rows number: 3
Enter columns number: 4
17      8      10     16
20      10     9      9
13      20     11     11

D:\my_work\test_ap\x64\Debug\test_ap.exe (process 1700) exited with code 0.
Press any key to close this window . . .
```

Опрацювання динамічних масивів у функціях

Динамічний масив, як і звичайний, можна передавати у функцію за допомогою *вказівника на масив, масива визначеного розміру або масива невизначеного розміру*. Найгнучкіший спосіб передачі масива в функцію є *вказівник на масив*. При передачі масива в функцію насправді передається не масив, а адреса його першого елемента.

Всередині функції багатовимірний масив сприймається як одновимірний, а його індекс формується з набору індексів багатовимірного масива.

Якщо потрібно виключити можливість зміни масива в функції, то його необхідно передавати в неї як константу.

Синтаксис заголовку функції, що отримує динамічний двовимірний масив-константу як одновимірний, має такий вигляд:

```
<тип_даних> <ім'я_функції> (const <тип_даних>*, [const]  
    <тип_даних>, ...);
```

де [] позначають необов'язковий елемент конструкції.

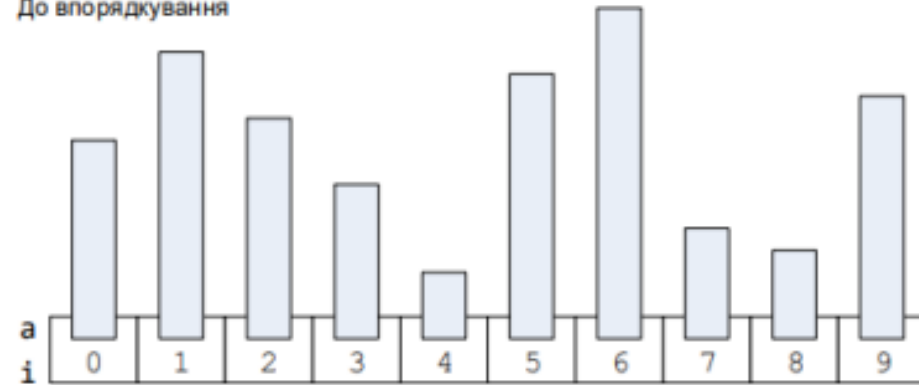
Тоді в програмі ця функція *повинна викликатись* так:

```
<ім'я_функції> (<ім'я_масива>, <ім'я_параметра>, ...);
```

Сортування масивів

Сортування або впорядкування масиву – це перестановка місцями його елементів так, щоб після перестановки елементи розміщувалися в певному порядку.

До впорядкування



Після впорядкування за зростанням



Всі способи впорядкування масивів поділяють на дві категорії:

- *методи внутрішнього сортування* -
- *методи зовнішнього сортування*

Всі методи внутрішнього сортування можна поділити на *два види*:

- *елементарні (прямі) методи*
- *вдосконалені методи.*

Є три основні способи прямого впорядкування

- 1) метод вибору;
- 2) метод вставки (включення);
- 3) метод обміну.

Метод вибору



a	9	3	7	5	8
i	0	1	2	3	4

a	9	3	7	5	8
i	0	1	2	3	4

min 3
imin 1

a	3	9	7	5	8
i	0	1	2	3	4

a	3	9	7	5	8
i	0	1	2	3	4

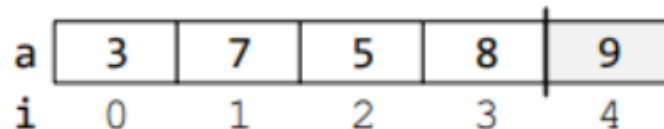
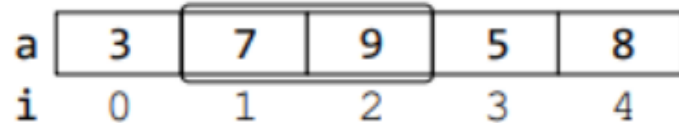
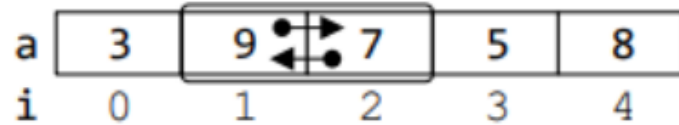
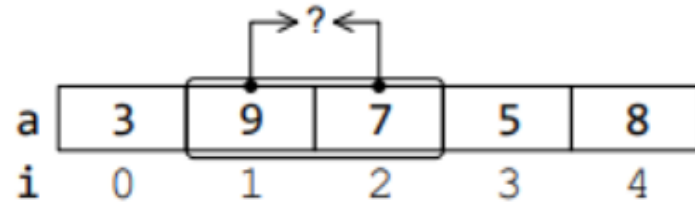
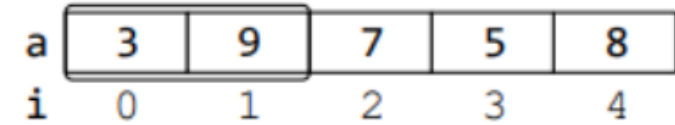
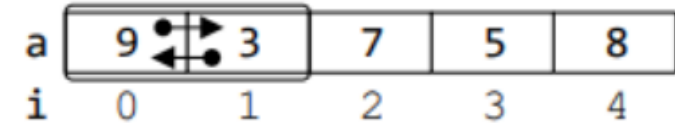
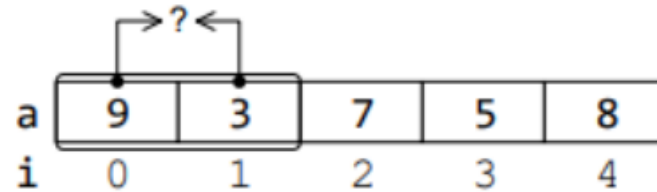
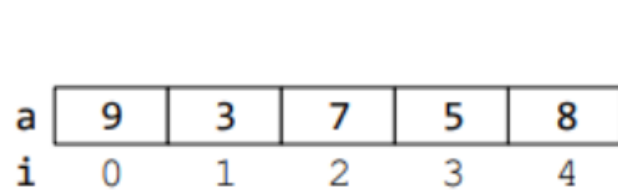
min 5
imin 3

a	3	5	7	9	8
i	0	1	2	3	4

Алгоритм методу вибору

1. Вважати весь масив – невідсортованою частиною
2. Поки невідсортована частина масиву містить більше одного елемента, виконувати дії:
 - 2.1. Знайти мінімальний елемент невідсортованої частини масиву і запам'ятати його індекс. ,
 - 2.2. Поміняти місцями мінімальний та вибраний на кроці 2.1 елементи масиву.
 - 2.3. Перемістити початок невідсортованої частини масиву на одну позицію праворуч

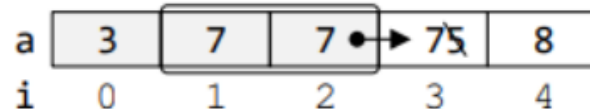
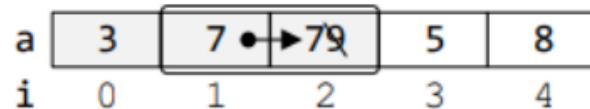
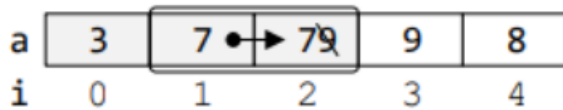
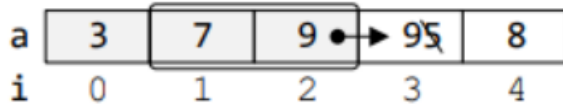
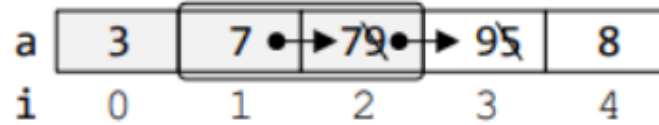
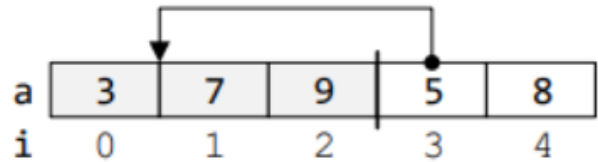
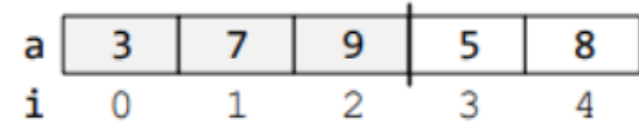
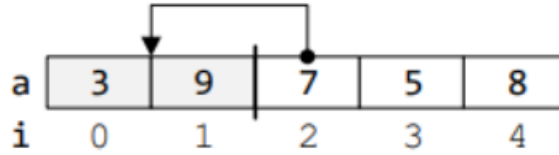
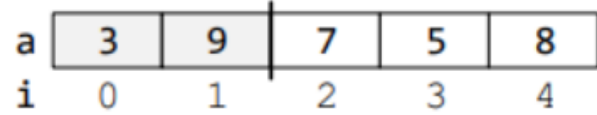
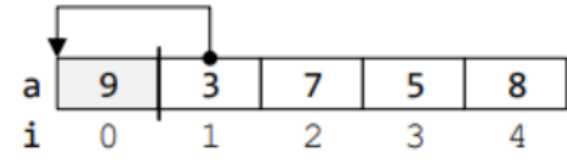
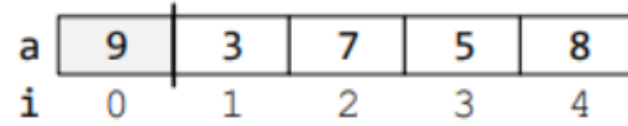
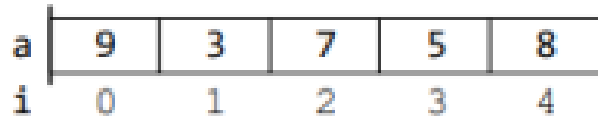
Метод обміну (бульбашки)



Алгоритм методу обміну (бульбашки).

1. Встановити лічильник ітерацій рівним одиниці.
2. Поки значення лічильника ітерацій менше кількості елементів масиву, виконувати дії:
 - 2.1. Встановити номер поточного елемента рівним нулю.
 - 2.2. Поки номер поточного елемента менший різниці кількості елементів масиву та значення лічильника ітерацій, виконувати дії:
 - 2.2.1. Якщо поточний елемент більший наступного – поміняти ці елементи місцями.
 - 2.2.2. Збільшити на одиницю номер поточного елемента (перейти до наступного елемента).
 - 2.3. Збільшити на одиницю лічильник ітерацій (перейти до наступної ітерації).

Метод вставки



Алгоритм методу вставки:

1. Вважати, що перший елемент масиву належить до впорядкованої частини, а індекс, з якого починається невідсортована частина, дорівнює 1
2. Поки невідсортована частина масиву містить хоча би один елемент, виконувати дії:
 - 2.1. Зберегти перший елемент невідсортованої частини масиву в допоміжній змінній.
 - 2.2. Визначити позицію вставки збереженого елемента у впорядковану частину масиву.
 - 2.3. Зсунути на одну позицію праворуч елементи відсортованої частини, які починаються з позиції вставки.
 - 2.4. Вставити збережений на кроці 2.1 елемент у знайдену позицію

Дякую за увагу

Лектор:
кандидат фіз.-мат. наук, доцент
Шаклеїна Ірина
iryna.o.shakleina@lpnu.ua
кафедра ІСМ, ІКНІ