Одновимірний масив

```
int mas[5];
```

Визначення масиву

Масив, це совокупність однотипних елементів, розташованих в пам’яті одним цілим фрагментом (куском). Тобто кожний наступний елемент, розташований відразу після попереднього.

Масиви бувають звичайні і динамічні. Для звичайних масивів необхідно на момент створення вказати кількість елементів. Розмір динамічних масивів можна вказати в момент роботи програми.

Спрощений синтаксис створення одновимірного масиву.

тип ім’я_масиву[розмір] [= {значення1, значення2,....., значення N}];

де тип – довільний тип;
розмір – ціла, додатня константа;
= {значення1, значення2,....., значення N} - ініціалізація;

[ ] – те що взяте в ці дужки означає, що не обов’язково.

Приклади створення одновимірного масиву

```
int mas1[5]; // масив із пя’яти цілих елементів
```

```
double mas2[4]={2.5, 3.3, 5.3, 2.3};//масив із чотирьох дійсних елементів
```

2,5	3,3	5,3	2,3

```
int mas2[5]={10,2}; // масив із пя’яти цілих елементів перші елементи
приймутьзначення відповідно 10 та 2 відповідно
```

10	2	0	0	0

```
int mas3[]={10, 2, 5, 4, 3, 7}; // розмір масиву визначить компілятор як 6
 // тобто по списку ініціалізації
```

10	2	5	4	3	7

Доступ до элемента масиву

Після створення масиву, є можливість «звернутися» до елемента масиву через індексацію (номер елементу у масиві). Індекс, це ціле число, яке визначає розташування елементу у масиві.

Індексація масиву розпочинається від нуля. Тому, якщо в масиві 5 елементів, початковий елемент з індексом нуль, останній, з індексом 4ри.

Синтаксис доступу до елементу масиву:

ім’я_масиву[індекс]

Доступ до елементів масиву виконується в двох напрямках:

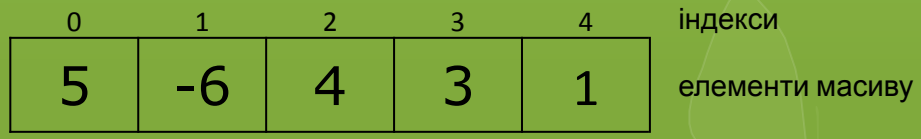
- в режимі читання,
- в режимі запису (зміна елемента масиву).

Синтаксис «Режим читання»

змінна = ім’я_масиву[індекс];

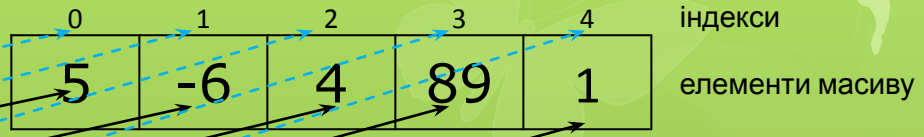
Синтаксис «Режим запису»

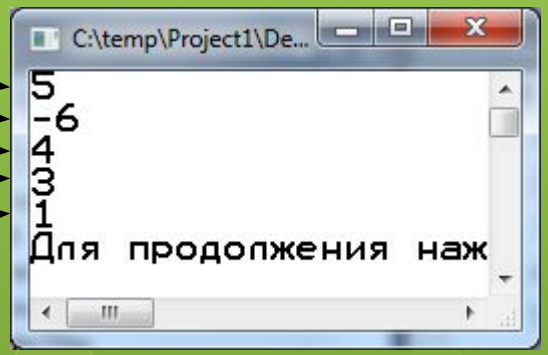
ім’я_масиву[індекс] = змінна;



Приклад доступу до елеметнів масиву

```
#include<iostream>
#include<stdlib.h>
using namespace std;
int main()
{
int mas[5];
mas[0] = 5;
mas[1] = -6;
mas[2] = 4;
mas[3] = 3;
mas[4] = 1;
cout << mas[0] << endl;
cout << mas[1] << endl;
cout << mas[2] << endl;
cout << mas[3] << endl;
cout << mas[4] << endl;
system("pause");
return 0;
}
```





Обхід усіх елементів масиву

Зазвичай обхід усіх елементів масиву виконується у циклі. Для цього необхідно створити цикл з деяким лічильником **i**, який прийме значення усіх можливих індексів елементів, тобто, якщо в маиві 5ть елементів, то **i** повине прийняти послідовно значення 0,1,2,3,4.

Якщо створити масив наступним чином

```
int mas[5];
```

тоді

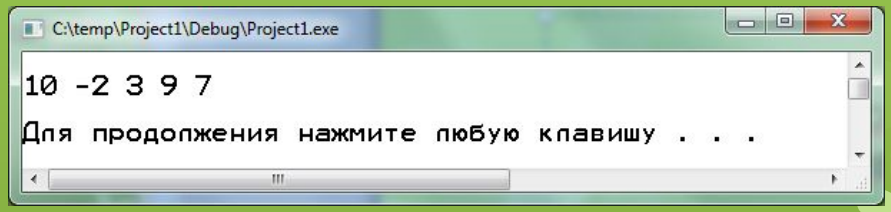
```
int mas[i]    - поточний елемент масиву
```

```
//заповнення деякого масиву розміру n у циклі
const int n = 5;
int mas[n];
for (int i = 0; i < n; ++i)
{
    mas[i] = деяке_значення;
}
```

Приклад

Створити масив цілих чисел та заповнити його випадковими елементами. Отриманий масив вивести на екран у рядок через пробіл

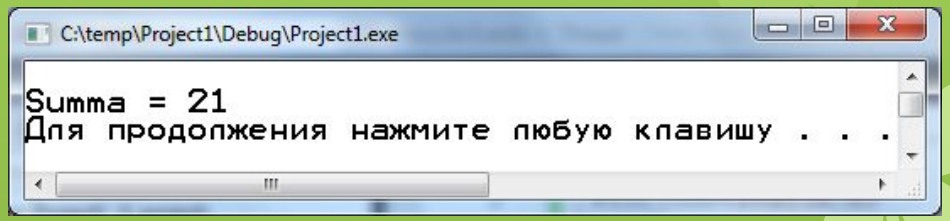
```
#include<iostream>
#include<stdlib.h>
using namespace std;
int main()
{
    const int n = 5;
    int mas[n];
    //заповнення масиву випадковими
    // числами від -10 до +10
    for (int i = 0; i < n; ++i)
    {
        mas[i] = rand() % 21 - 10;
    }
    cout << endl;
    //вивід масиву на екран
    for (int i = 0; i < n; ++i)
    {
       cout << mas[i] << " ";
    }
    cout << endl<<endl;
    system("pause");
    return 0;
}
```



Приклад 2

Знайти суму усіх елементів масиву

```
#include<iostream>
#include<stdlib.h>
using namespace std;
int main()
{
    const int n = 5;
    int mas[n] = {5, 3, 7, 2, 4};
    int s = 0;
    for (int i = 0; i < n; ++i)
    {
       s+=mas[i];
    }
    cout <<"\nSumma = "<<s<< endl;
    system("pause");
    return 0;
}
```



Знаходження екстремумів у масиві (min/max)

Головна ідея знаходження мінімально/максимального елементу масив:

На першому кроці вважати, що початковий елемент мінімальний/максимальний.

Порівняти цей елемент з усіма іншими елементами масиву, якщо зустрінеться елемент менший/більший, чим зберігається, тоді перевизначити значення поточного мінімального/максимального.

Алгоритм знаходження значення мінімального (максимального) елемента масиву.

1) Створити змінну min/max в яку записати значення початкового елементу масиву min = mas[0];/max = mas[0];

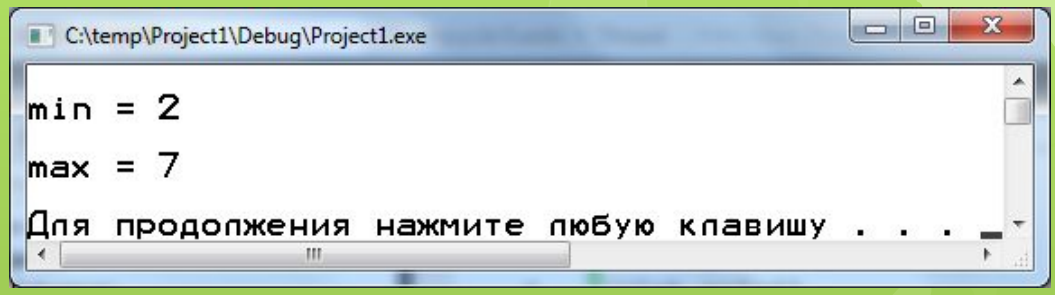
2) В циклі порівняти поточний елемент масиву mas[i] з min/max.

Якщо зустрівся елемент меньший/більший, тоді перевизначити min/max: min = mas[i]; /max = mas[i].

Приклад 3

Знайти мінімальний та максимальний елемент одновимірного масиву

```
#include<iostream>
#include<stdlib.h>
using namespace std;
int main()
{
    const int n = 5;
    int mas[n] = {5, 3, 7, 2, 4};
    int min = mas[0];
    int max = mas[0];
    for (int i = 1; i < n; ++i)
    {
       if (mas[i] < min) min = mas[i];
       if (mas[i] > max) max = mas[i];
    }
    cout << "\nmin = " << min << endl;
    cout << "\nmax = " << max << endl << endl;
    system("pause");
    return 0;
}
```



Знаходження індексу мінімального/максимального елементу

В попередньому прикладі було знайдено значення мінімального та максимального елементів, проте попередній алгоритм не надає інформації про те де вони розташовані. Наприклад, якщо поставити задачу поміняти місцями екстремуми, нам необхідно місце розташування (індекси) цих елементів.

Модифікуємо попередній алгоритм: будемо зберігати не значення минимального/максимального, а їх індекси.

Якщо створити змінні imin/imax для зберігання індесів мінімального/максимального елементу, тоді

mas[imin] - поточний мінімальний

mas[imax] - поточний максимальний

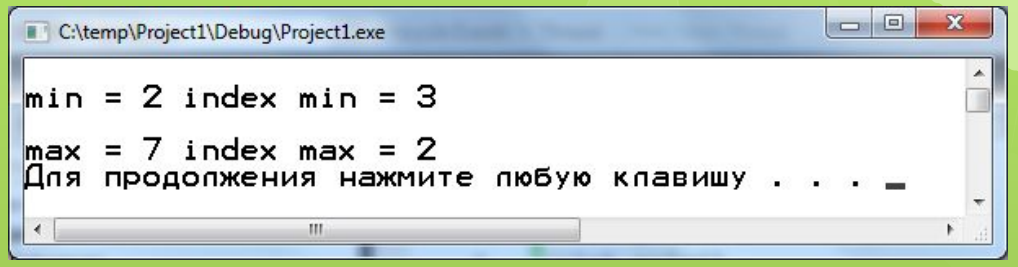
Алгоритм:

- 1) Створити змінну imin/imax в яку записати значення початкового індексу 0: imin = 0 /imax = 0
- 2) В циклі порівняти поточний елемент масиву mas[i] з mas[imin]/ mas[imax].
- Якщо зустрівся елемент меньший/більший, тоді перевизначити збережений індекс imin/imax: imin = i / imax = i.

Приклад 4

Знайти індекс мінімальний та максимальний елемент одновимірного масиву

```
#include<iostream>
#include<stdlib.h>
using namespace std;
int main()
{
    const int n = 5;
    int mas[n] = {5, 3, 7, 2, 4};
    int imin = 0;
    int imax = 0;
    for (int i = 0; i < n; ++i)
    {
       if (mas[i] < mas[imin]) imin = i;
       if (mas[i] > mas[imax]) imax = i;
    }
    cout << "\nmin = " << mas[imin]
         << " index min = " << imin<<endl;
    cout << "\nmax = " << mas[imax]
         << " index max = " << imax << endl;
    system("pause");
    return 0;
}
```



Знаходження адреси мінімального/максимального елементу

Розглянемо третій варіант знаходження мінімального/максимального елементу. На відміну від попереднього, будемо зберігати не індес а адресу поточного мінімального/максимального.

Модифікуємо попередній алгоритм: будемо зберігати не значення минимального/максимального, а їх адреси.

Якщо створити покажчики pmin/pmax для зберігання адреси мінімального/максимального елементу, тоді

*pmas - поточний мінімальний

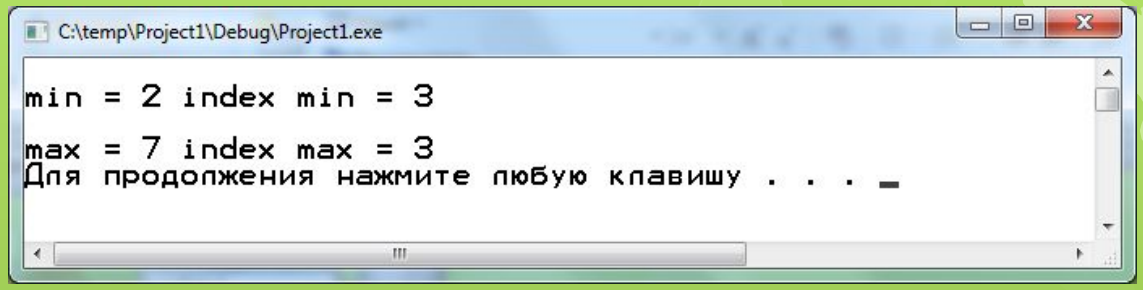
*pmas - поточний максимальний

Алгоритм:

- 1) Створити покажчики pmin/zmax в яку записати значення початкового індексу 0: pmin = &mas[0] / pmax = &mas[0]
- 2) В циклі порівняти поточний елемент масиву mas[i] з *pimin/ *pmas.
- Якщо зустрівся елемент меньший/більший, тоді необхідно зберегти нові адреси pmin/pmax: pmin = &mas[i] / pmax = &mas[i].

Приклад 5

Знайти адреси мінімального та максимального елементів одновимірного масиву




```
#include<iostream>
#include<stdlib.h>
using namespace std;
int main()
{
    const int n = 5;
    int mas[n] = { 5, 3, 7, 2, 4 };
    int* pmin = &mas[0]; //int* pmin = mas;
    int* pmax = &mas[0];
    for (int i = 0; i < n; ++i)
    {
       if (mas[i] < *pmin) pmin = &mas[i];// = mas+i;
       if (mas[i] > *pmax) pmax = &mas[i];// = mas+i;
    }
    cout << "\nmin = " << *pmin
       << " index min = " << pmin-mas << endl;
    cout << "\nmax = " << *pmax
       << " index max = " << pmin-mas << endl;
    system("pause");
    return 0;
}
```

Сортування одновимірного масиву

Критерій відсортованості масиву: Масив вважається відсортованим, якщо для любих його сусідів виконується вибраний критерій.

Наприклад, якщо кожний наступний елемент буде більший за попередній то масив буде відсортований за зростання, наприклад, - 9 -5 3 5 7 89.

Усі алгоритми сортування включають в себе дві загальні процедури:

1) Порівняння деяких двох елементів із масиву
2) Якщо для вибраних елементів не виконується критерій сортування, необхідно їх переставити місцями.

Порівняння та перестановка елементів виконується до тих пір, поки для усіх елементів масиву не виконається «Критерій відсортованості масиву».

Кількість порівнянь, та які елементі необхідно порівнювати залежить від вибраного алгоритму сортування.

Розглянемо найпростійший алгоритм: Метод бульбашки

Сортування одновимірного масиву методом «бульбашки»

Головна ідея алгоритму методи бульбашки включає в себе дві процедури

1) «Порівняння усіх сусідів»
2) Повторення процедури «Порівняння усіх сусідів» n-1 раз, де n – кількість елементів у масиві

Тепер розгляномо з чого складається «Порівняння усіх сусідів»

Алгоритм «Порівняння усіх сусідів», на прикладі сортуваня за зростання (Критерій відсортованості: Наступний елемент повинен бути меньший від поточного)

По парно порівняємо усі елементи у масиві, якщо поточний елемент mas[i], тоді наступний елемент mas[i+1].

1) Порівнуємо mas[i+1] <mas[i]
2) Якщо нерість 1) не виконується міняємо ці елементі місцями.

Приклад 5

Відсортувати одновимірний масив методом бульбашки

```
#include<iostream>
#include<stdlib.h>
using namespace std;
int main()
{
const int n = 5;
int mas[n] = { 5, 3, 7, 2, 4 };
for (int k = 0; k < n-1; ++k) //повторення порівнання усіх сусідів
for (int i = 0; i < n-1-k; ++i)//процедура порівняння усіх сусідів
{
if (mas[i + 1] < mas[i])
{//перестановка елементів за правилом трьох стаканів
int c = mas[i + 1];
mas[i + 1] = mas[i];
mas[i] = c;
}
}
//виведення результату на екран
for (int i = 0; i < n; ++i)
{
cout << mas[i] << " ";
}
cout << endl;
system("pause");
return 0;
}
```