

Программирование на языке C++

Лекция 6

Массивы. Одномерные массивы

Что такое массив

- Массив – это множество элементов;
- Все элементы массивы одного типа;
- Элементы расположены в памяти непрерывно, друг за другом;
- Доступ к элементам массива осуществляется с помощью общего для них имени.

Виды массивов

- Статические
 - Одномерные
 - Многомерные
- Динамические
 - Одномерные
 - Многомерные

Статические одномерные массивы

Объявление массива

```
тип_элемента имя_массива[размер_массива];
```

```
int num[10];
```

```
int a[10], b[2];
```

```
const int size = 28;
```

```
char letter[size]; // Только если size константа
```

```
#define SIZE 10 // Везде в коде SIZE заменять на 10
```

```
double array[SIE];
```

Инициализация массива

```
int trash[10]; // Значения массива не известны (мусор)
```

```
// Размер 10
```

```
int digit[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

```
// В C++11 знак '=' можно не писать
```

```
char hello[] {'П', 'р', 'и', 'в', 'е', 'т'};
```

```
int zero[5] = {1}; // Все элементы, кроме первого нули
```

```
bool flags[3] = {true, false, false};
```

```
int num[2] = {0.5, 1.3}; // Ошибка: int уже чем double
```

```
double num2[2] = {1, -3}; // тут всё нормально
```

Работа со статическим одномерным массивом

```
int array[] = { 5, -12, -12, 9, 10, 0, -9};
```

```
// Массив всегда начинается с нулевого индекса
```

```
int num = 1 + array[6]; // num = -8
```

```
cout << array[0]; // 5
```

```
array[1] = 0; // { 5, 0, -12, 9, 10, 0, -9};
```

```
array[2] = -array[2]; // { 5, 0, 12, 9, 10, 0, -9};
```

```
// Выход за границы массива не проверяется
```

```
array[-1] = 3;
```

```
array[100] = 3;
```

Работа со статическим одномерным массивом

```
int src[] = { 5, -12, -12, 9, 10, 0, -9};
```

```
int dest[7];
```

```
dest[] = { 5, -12, -12, 9, 10, 0, -9}; // Ошибка
```

```
dest = src; // Ошибка
```

```
// Копируем значения из src в dest
```

```
for (int i = 0; i < 7; i++){  
    dest[i] = src[i];  
}
```

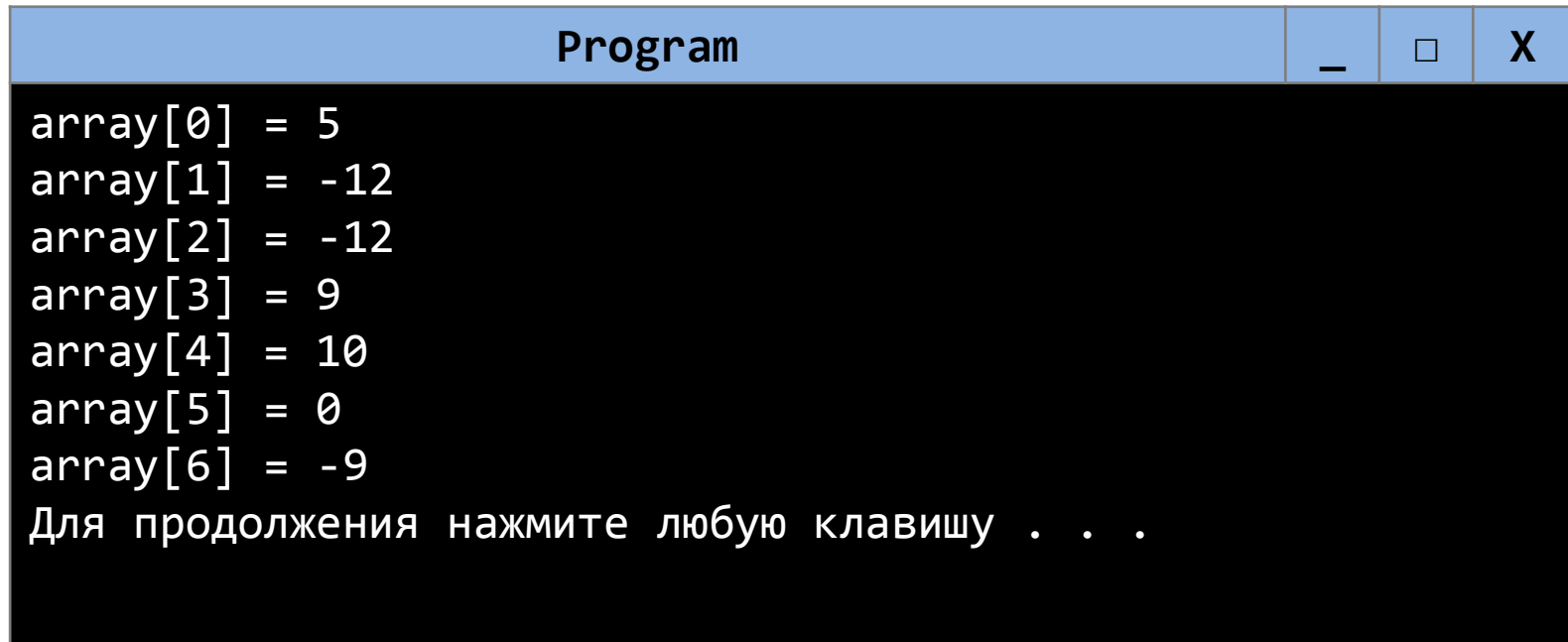

Вывод массива на экран

```
#include <iostream>
```

```
int main(){  
    const int size = 7;  
    int array[size] = { 5, -12, -12, 9, 10, 0, -9};  
    for (int i = 0; i < size; i++){  
        std::cout << "array[" << i << "] = ";  
        std::cout << array[i];  
        std::cout << std::endl;  
    }  
    system("pause");  
    return 0;  
}
```

Вывод массива на экран

```
int array[7] = { 5, -12, -12, 9, 10, 0, -9};
```



The screenshot shows a window titled "Program" with a black background and white text. The text displays the contents of an array: array[0] = 5, array[1] = -12, array[2] = -12, array[3] = 9, array[4] = 10, array[5] = 0, and array[6] = -9. Below the array values, there is a prompt in Russian: "Для продолжения нажмите любую клавишу . . .". The window has a standard title bar with a minus sign, a maximize button, and a close button (X).

```
Program
```

```
array[0] = 5  
array[1] = -12  
array[2] = -12  
array[3] = 9  
array[4] = 10  
array[5] = 0  
array[6] = -9  
Для продолжения нажмите любую клавишу . . .
```

Как устроен одномерный статический
массив

Размещение массива в памяти

```
double a = 3.1;
int arr[4] = {-1, 7, 3, 5};
double b = 2.0;
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a = 3.1								arr[0] = -1				arr[1] = 7			
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
arr[2] = 3				arr[3] = 5				b = 2.0							

Как это видит компилятор

```
double a = 3.1;  
int arr[4] = {-1, 7, 3, 5};  
double b = 2.0;
```

Имя	Тип	Размер, байт	Адрес
a	double	8	2293400
arr	int [4]	16	2293408
b	double	8	2293424

Поведение массива в коде

Имя массива в коде эквивалентно константному указателю на тип элемента массива.

Значение этого указателя зафиксировано на элементе массива с индексом нуль.

```
// Массив из 4х элементов типа double
double array[4];
// Константный указатель на double
double (*const p) = &array[0];
```

Исключения: операторы **sizeof** и **&**(взятие адреса), ...

Имя массива как указатель

```
double a = 3.1;  
int arr[4] = {-1, 7, 3, 5};  
double b = 2.0;  
int *p = &arr[0];
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a = 3.1								arr[0] = -1				arr[1] = 7			
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
arr[2] = 3				arr[3] = 5				b = 2.0							

```
int *p2 = (p + 1); // int *p2 = (arr + 1);
```

Оператор []

```
int arr[4] = {-1, 7, 3, 5};  
int *p = arr;
```

```
cout << p[2];      // 3
```

```
cout << *(p + 2);  // 3
```

```
cout << arr[2];     // 3
```

```
cout << *(arr + 2); // 3
```


Что это даёт

```
int arr[4] = {-1, 7, 3, 5};
```

```
int *p = arr;
```

```
while(p < arr+4) cout << *(p++);
```

```
p = arr+3; // Последний элемент массива
```

```
cout << p[0] << p[-1] << p[-2] << p[-3];
```

```
p = arr-5;
```

```
cout << p[5]; // arr[0]
```

Массив как аргумент функции

```
#include <iostream>
int sum(int array[10], int size); // Прототип
int sum(int array[], int size);   // Прототип
int sum(int *array, int size);    // Прототип

int main(){
    int arr[4] = {1, 2, 3, 4};
    std::cout << sum(arr, 4);    // Передача по имени
}

int sum(int *array, int size){
    int res = 0;
    while(--size >= 0) res += array[size];
    return res;
}
```

Массив как аргумент функции

- В функцию массив передаётся по имени;
- Значения массива не копируются, копируется указатель;
- Внутри функции не возможно отличить передан обычный указатель или массив;
- Возврат массива из функции происходит тоже через указатель

Динамические одномерные массивы

Объявление

```
тип_элемента *имя = new тип_элемента[размер];  
int *array = new int[10];
```

Созданный динамически массив обязательно
удалить, когда он уже не нужен

```
delete[] array;
```

Использование

Использование динамически объявленных массивов полностью идентично использованию статических массивов

Возвращение одномерного массива из функций

// Так делать нельзя, массив `array` существует только внутри функции `fill_zerro`

```
int* fill_zerro(){  
    int array[10] = {0};  
    return array;  
}
```

```
int main(){  
    int* arr = fill_zerro();  
    return 0;  
}
```

Возвращение одномерного массива из функций

// Массив `array` существует независимо от функции `fill_zerro`

```
int* fill_zerro(){  
    int *array = new int[10] {0};  
    return array;  
}  
  
int main(){  
    int* arr = fill_zerro();  
    delete[] arr; // Не забываем удалить  
    return 0;  
}
```