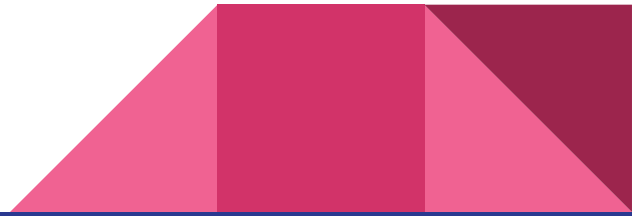
Лекция 10

Статические и динамические массивы. Класс Array.

Массив

Массив — это структура данных, представляющая собой набор переменных одинакового типа, имеющих общее имя.

Массивы удобно использовать для хранения однородной по своей природе информации, например, таблиц и списков.



Пример инициализации массива

Возьмем группу студентов из десяти человек. Создадим массив, в котором будут храниться фамилии всех студентов.

```
string students[10] = {
    "Иванов", "Петров", "Сидоров",
    "Ахмедов", "Ерошкин", "Выхин",
    "Андеев", "Вин Дизель", "Картошкин", "Чубайс"
};
```

Описание синтаксиса

Массив создается почти так же, как и обычная переменная. Для хранения десяти фамилий нам нужен массив, состоящий из 10 элементов. Количество элементов массива задается при его объявлении и заключается в квадратные скобки.

Массив типа **int** из 10 элементов описывается с помощью адреса его первого элемента и количества байт, которое может вместить этот массив. Если для хранения одного целого числа выделяется 4 байта, то для массива из десяти целых чисел будет выделено 40 байт.

Вывод первого элемента массива

```
#include <iostream>
#include <string>
int main()
{       std::string students[10] = {
        "Иванов", "Петров", "Сидоров",  "Ахмедов", "Ерошкин", "Выхин",
        "Андеев", "Вин Дизель", "Картошкин", "Чубайс"    };
    std::cout << students[0] <<  std::endl;
    return 0;     }
```

Вывод элементов массива через цикл

```
#include <iostream>
#include <string>
int main()
{    std::string students[10] = {
        "Иванов", "Петров", "Сидоров", "Ахмедов", "Ерошкин", "Выхин",
        "Андеев", "Вин Дизель", "Картошкин", "Чубайс"    };
    for (int i = 0; i < 10; i++) {   std::cout << students[i] << std::endl;   }
    return 0;   }
```

Объявление массива без инициализации

string students[10];

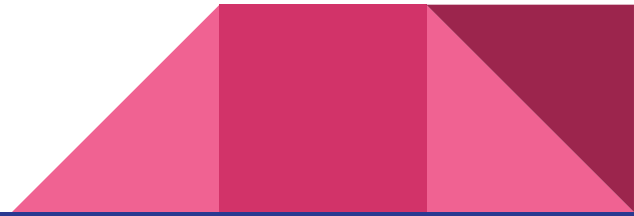
// или

string teachers[5];

Элементы такого массива обычно содержат в себе «мусор» из выделенной, но еще не инициализированной, памяти. Некоторые компиляторы, такие как GCC, заполняют все элементы массива нулями при его создании.

Статический массив

При создании статического массива, для указания его размера может использоваться только константа. Размер выделяемой памяти определяется на этапе компиляции и не может изменяться в процессе выполнения.

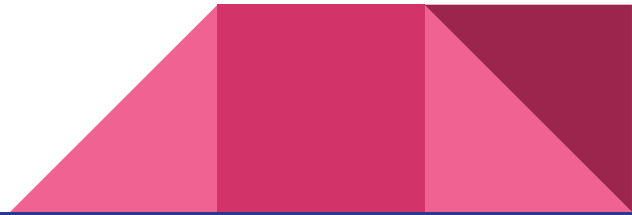


Динамический массив

При создании динамического массива выделяется определенное количество памяти для массива, значение которого изначально неизвестно.

Например, необходимо создать динамический массив из N элементов, где значение N задается пользователем.

Синтаксис выделения памяти для массива имеет вид **указатель = new тип [размер]**. В качестве размера массива может выступать любое целое положительное значение.



Создание динамического массива

```
#include <iostream>

using namespace std;

int main()

{    int num;                                  // размер массива

    cout << "Enter integer value: ";

    cin >> num;                               // получение от пользователя размера массива

    int *p_darr = new int[num];               // Выделение памяти для массива

    for (int i = 0; i < num; i++) {           // Заполнение массива и вывод значений его элементов

        p_darr[i] = i;

        cout << "Value of " << i << " element is " << p_darr[i] << endl;    }

    delete [] p_darr;                         // очистка памяти

    return 0;      }
```

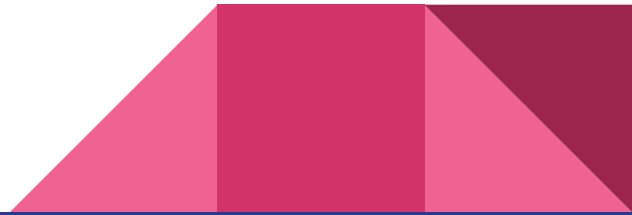
Многомерные массивы

Кроме одномерных массивов в C++ есть многомерные. Элементы таких массивов сами в свою очередь являются массивами, в которых также элементы могут быть массивами. Например, определим двухмерный массив чисел:

```
int numbers[3][2] = { {1, 2}, {4, 5}, {7, 8} };
```

Чтобы обратиться к элементам вложенного массива, потребуется два индекса:

```
numbers[1][0] = 12;
```



Класс Array

Описывает объект, управляющий последовательностью из элементов **N** типа **T**. Последовательность хранится как массив **T** в объекте **array<T, N>**.

```
template <class T, std::size_t N>
class array;
```

Класс Array

У этого типа есть конструктор по умолчанию **array()** и оператор присваивания по умолчанию **operator=**. Тип удовлетворяет требованиям для **aggregate**. Поэтому объекты типа **array<T, N>** можно инициализировать с помощью агрегатного инициализатора. Например, примененная к объекту директива

array<int, 4> ai = { 1, 2, 3 };

создает объект **ai**, содержащий четыре целочисленных значения, инициализирует первые три элемента как 1, 2 и 3 соответственно и инициализирует четвертый элемент как 0.