Массивы в C++

На C я могу просто делать ошибки, на C++ я могу их наследовать!

Массивы в C++

Создается массив по следующей схеме:

тип имя[размер];

Где тип — это тип элементов массива, имя — это допустимый и уникальный в данной области видимости идентификатор, а размер — это положительный литерал, переменная или константа целого типа.

Массивы в C++

Можно не указывать размер массива (оставив пустые квадратные скобки после его имени), но тогда необходимо сразу перечислить все его элементы (инициализировать массив), в этом случае размер автоматически вычислит компилятор. Примеры корректного объявления массивов:

```
int mas1[4];
int mas2[] = {3,-7,9,1200,-713};
float mas3[400];
const int n = 173;
double mas4[n];
```

Массивы в C++

- Массив объявленный как char mas[128]; будет занимать в памяти — 128 байт (столько же заняли бы 128 разных переменных типа char, каждая из которых занимает по байту). При этом размер массива не может быть сколь угодно большим (например, если массив объявлен не как глобальный, то его максимально допустимый размер зависит от доступного стека).

Массивы в C++

Массив **mas3** займет в памяти 1600 байт.

Сколько места в памяти займут остальные из объявленных в примере массивов?

```
int mas1[4];
int mas2[] = {3,-7,9,1200,-713};
float mas3[400];
const int n = 173;
double mas4[n];
```

Массивы в C++

В неинициализированном массиве (по аналогии с неинициализированной переменной) будут храниться заранее неизвестные значения (какой-то «мусор», ранее записанный другими программами или даже вашей программой в выделяемую для объявленного массива

Массивы в C++

Чтобы обратиться к какому-то из элементов массива для того, чтобы прочитать или изменить его значение, нужно указать имя массива и за ним индекс элемента в квадратных скобках.

Элемент массива с конкретным индексом ведёт себя также, как переменная.

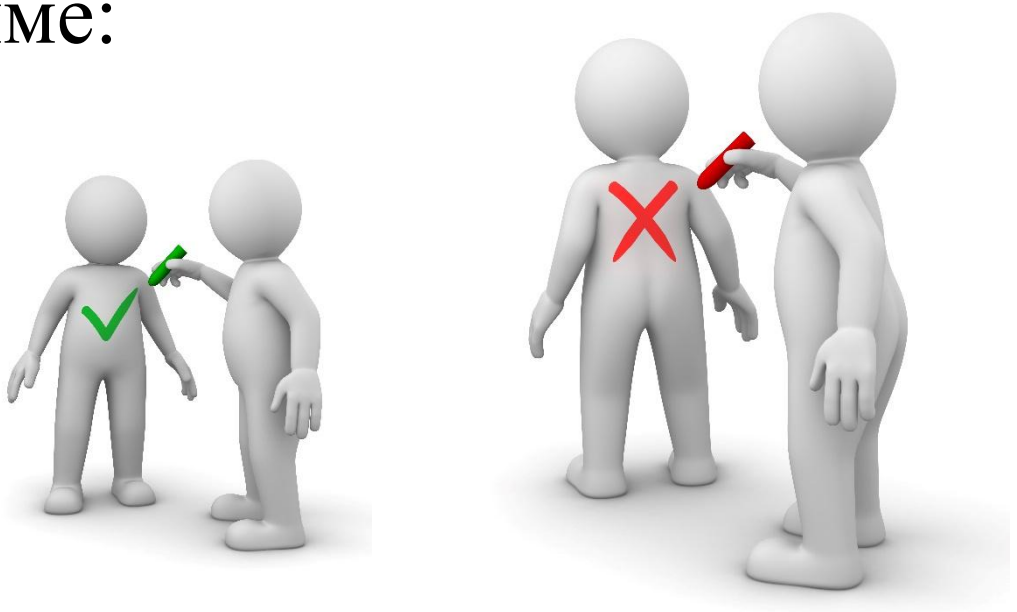
Например, чтобы вывести значения первого и последнего элементов массива **mas1** надо написать в программе:

```
cout << mas1[0];
cout << mas1[3];
```

Массивы в C++

А чтобы присвоить новые значения (10, 20, 30, 40) всем элементам массива, потребуется написать в программе:

- mas1[0] = 10;
- mas1[1] = 20;
- mas1[2] = 30;
- mas1[3] = 40;



Массивы в C++

Уже из последнего примера видно, что для того, чтоб обратиться ко всем элементам массива, приходится повторять однотипные действия.

Для многократного повторения подобных операций используются циклы. Соответственно, мы могли бы заполнить массив нужными элементами с помощью цикла:

```
for(int i=0; i<4; i++)
  {
    mas1[i] = (i+1) * 10;
  }
```

Массивы в C++

А после этого несложно вывести все элементы массива на экран:

```
for(int i=0; i<4; i++)
  {
    cout << mas1[i] << '  ';
  }
```

Пример программы

```
#include <iostream>
using namespace std;
int main() {
   cout << "Укажите размер массива: ";
   int n;
   cin >> n;
   const int dim = n;
   int arr[dim];
   for(int i=0; i<dim; i++)
     {
       arr[i] = i+1;
     }
   for(int i=dim-1; i>=0; i--)
    {
       cout << arr[i] << ' ';
    }
}
```

Генерация псевдослучайных чисел в C++

Что бы такого придумать…

Генерация псевдослучайных чисел в C++

В стандартной библиотеке C++ (в заголовочном файле **cstdlib**) существует функция rand(), которая при каждом вызове возвращает псевдослучайное целое число в диапазоне от 0 до константы RAND_MAX (обычно она равна 32767, но её значение зависит от настроек среды и, соответственно, может изменяться, самый простой способ избавиться от сомнений — вывести значение этой константы на экран).

Генерация псевдослучайных чисел в C++

Если создать и запустить программу, которая выводит два случайных числа на экран несколько раз, то можно увидеть, что от запуска к запуску числа повторяются, хотя до первого запуска, конечно, нельзя было предсказать эту пару значений.

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
 cout << "Выводим 2 случайных значения от 0 до " << RAND_MAX << endl;
 cout << rand() << ' ' << rand();
 return 0;
}
```

Генерация псевдослучайных чисел в C++

Функция rand(), на самом деле, выбирает числа из последовательности значений, вычисленных по специальному алгоритму, где базой для построения этой последовательности является некоторый числовой аргумент (обычно называемый seed — «зерно» с англ.).

Значение этого аргумента неизменно между запусками программы, но изменится, если, например, перезагрузить компьютер.

Генерация псевдослучайных чисел в C++

Иметь одинаковые наборы при каждом запуске удобно, например, в процессе тестирования. Но для реальной программы нужно, чтобы последовательность псевдослучайных чисел менялась даже при двух последовательных запусках программы. Для её перемещивания существует функция **srand(seed)**, где *seed* — это некоторое значение типа unsigned int.

Соответственно, если в качестве аргумента в эту функцию передавать разные значения при каждом запуске программы, то и наборы получаемые с помощью rand() — будут разными.

Генерация псевдослучайных чисел в C++

Традиционно в качестве аргумента для функции srand() используют текущее значение времени в формате **UNIXTIME** (количество секунд, прошедших с 1 января 1970 года).

Его можно получить с помощью функции time (NULL), которая также является частью стандартной библиотеки и описана в заголовочном файле ctime.

Генерация псевдослучайных чисел в C++

Предыдущий пример можно легко модернизировать до такого состояния, чтобы числа не повторялись между запусками:

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
int main()
{
 cout << "Выводим 2 случайных значения от 0 до " << RAND_MAX << endl;
 srand(time(NULL));
 cout << rand() << ' ' << rand();
 return 0;
}
```

Генерация псевдослучайных чисел в C++

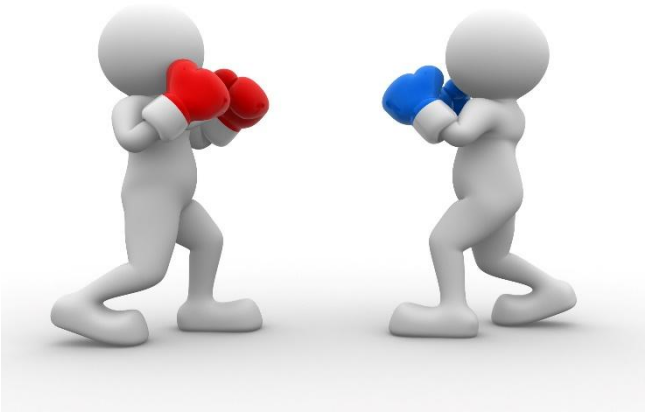
Отметим, что перемешивать последовательность псевдослучайных чисел нет смысла чаще одного раза в процессе исполнения программы.

В реальных задачах требуется получать числа из определенных промежутков (не от 0 до RAND_MAX).

Генерация псевдослучайных чисел в C++

Проблема решается с использованием пары арифметических операций. Для того, чтобы сузить промежуток до [0;n−1] достаточно применить к значению функции rand() операцию остатка от деления на n (оператор «%»).

То есть выражение rand()%n будет всегда возвращать псевдослучайное число из отрезка от 0 до n−1 (заметим, что целых чисел там ровно n штук), при это распределение получаемых значений по классам вычетов будет достаточно равномерным (а если RAND_MAX+1 кратно n, то полностью равномерным).



Генерация псевдослучайных чисел в C++

Когда требуется получить промежуток начинающийся не от 0, то результат функции просто сдвигают в положительном или отрицательном направлении, соответственно, прибавляя или вычитая нужное значение. То есть выражение rand()%n+a будет всегда возвращать псевдослучайное число из отрезка [a;a+n−1] (в нём тоже ровно n целых чисел).

Итак, чтобы получить псевдослучайное число из нужного отрезка, нужно сузить значение функции rand() до длины этого отрезка и сдвинуть на значение левого конца отрезка.

Генерация псевдослучайных чисел в C++

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
int main()
{
	int a, b;
	cout << "Введите левый конец отрезка "; cin >> a;
	cout << "Введите правый конец отрезка "; cin >> b;
	srand(time(NULL));
	int rnd = rand()%(b-a+1)+a;
	cout << "Псевдослучайное число из отрезка [" << a << ';' << b <<"]: " <<
rnd;
 }
```

Задачи

1. Вывести на экран 10 чисел.
2. Вывести на экран 5 натуральных чисел из диапазона от 50 до 75.
3. Вывести на экран 5 вещественных чисел из диапазона от 5 до 8.