top
КОМПЬЮТЕРНАЯ АКАДЕМИЯ

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C++

Урок № 6

Массивы

Содержание

Материалы урока прикреплены к данному PDF-файлу. Для доступа к материалам, урок необходимо открыть в программе Adobe Acrobat Reader.

1. Необходимость группировки данных

Сегодня мы поговорим с вами на тему хранения данных. На одном из первых занятий мы узнали о существовании переменной и определили ее как отрезок оперативной памяти для размещения информации. Несомненно, что нормальная программа не может существовать без переменных, однако, порой простые переменные не решают проблемы оперирования данными. Представим себе, что нам необходимо посчитать среднюю оценку по группе. Очень легко! Для этого мы в цикле запросим все оценки и прибавим их к переменной для суммы. А после выполнения цикла данную переменную разделим на количество студентов. Все готово. А что если нам вдруг после этого нужно найти еще и максимальную оценку по группе. Сможем ли мы это сделать?! К сожалению, нет. А, дело все в том, что каждая из переменных, способна одновременно хранить лишь один элемент информации. И каждый раз в цикле она перезаписывала предыдущее значение оценки. Будет весьма неудобно создавать для каждого элемента переменную. А, что если требуется работать со многими сотнями элементов? Задача очень быстро становится невыполнимой. Согласитесь, создавать несколько сотен переменных — безумие.

Как же решить такую казалось бы непростую задачку?! В нашем случае решением являются, так называемые массивы. Рассмотрим определение и особенности.

Понятие массива

Массив — это набор однотипных данных объединенный общим именем. Представим себе массив в виде пассажирского поезда, состоящий из вагонов. Предположим, что необходимо создать программу, которая хранит в себе информацию о загруженности поезда по вагонам (рис. 1).

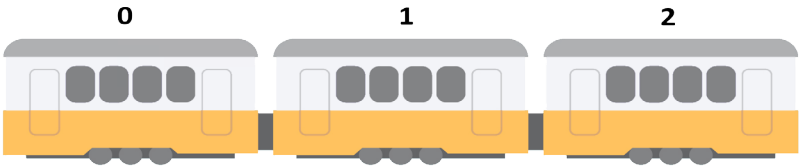

Рисунок 1

Давайте попробуем создать такой массив. Для этого сначала разберемся с основными правилами.

Синтаксис объявления массива

```
тип_данных имя_массива[количество_элементов];
```

1. **Тип_данных** — любой из существующих, известных вам типов данных. Именно этим типом будет обладать каждый элемент массива.
2. **Имя_массива** — любое имя, которое подчиняется «правилам имен переменных» (эти правила мы рассматривали с Вами в первом уроке).
3. **Количество_элементов** — число элементов в массиве. На данном месте должно находиться — положительное целочисленное константное значение. Таким значением может быть — либо целочисленный литерал, либо константная целочисленная переменная.

Примечание. *Обратите внимания, что количество элементов массива должно быть определенно на этапе создания программы. Другими словами, задать размерность такого массива в зависимости от какого-то условия или по решению пользователя невозможно. Это приведет к ошибке на этапе компиляции.*

А теперь представим себе небольшой поезд, в составе которого всего лишь 3 вагона. Загруженность вагона характеризуется количеством занятых уже мест. То есть каждая ячейка массива будет хранить количество зарезервированных мест в вагоне. Поэтому **int** как тип данных нам подойдет лучше всего.

```
int train[3];
```

Такая запись указывает, что в памяти было выделено три ячейки, каждая из которой занимает 4 байта (рис. 2).

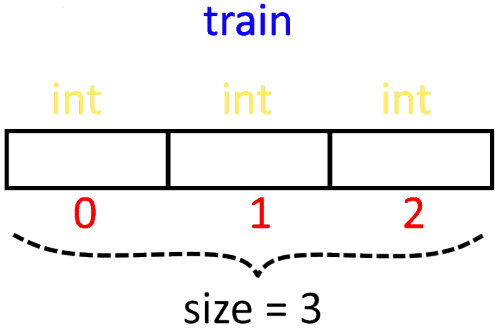

Рисунок 2

Как видим, каждая такая переменная в массиве является самостоятельной единицей под названием — элемент.

А каждый элемент имеет свой порядковый номер — индекс. По индексу можно обращаться к конкретному элементу массива. Нумерация элементов в массиве начинается с нуля.

Таким образом, чтобы заполнить массив значениями мы должны знать его имя и номер нужной ячейки.

```
train[0] = 3;
train[1] = 1;
train[2] = 4;
```

Эти строки указывают, что в первом вагоне (нулевой индекс) занято три места, во втором (первый индекс) — одно, а в третьем (второй индекс) — четыре (рис. 3).

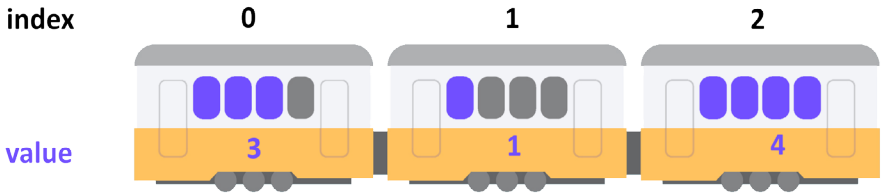

Рисунок 3

Давайте рассмотрим другой пример. В группе учится 12 студентов. Каждый студент получает оценку за выполненное домашнее задание. Необходимо вывести на экран оценку выбранного студента.

Сначала определяем, сколько ячеек нам нужно (двенадцать студентов = двенадцать оценок).

```
const int N = 12;
```

Допустим, что оценки выставлены в алфавитном порядке расположения студентов в списке. Более того, оценки

могут содержать целое значение от 1 до 12. Сформируем целочисленный массив с оценками и заполним его.

```
// create a digit value array of twelve elements
int mark[N];

// fill the array
mark[0] = 10; // 10 - mark of the first student
mark[1] = 7; // 7 - mark of the second student
mark[2] = 11; // 11 - mark of the third student
mark[3] = 10; // 10 - mark of the fourth student
mark[4] = 9; // 9 - mark of the fifth student
mark[5] = 8; // 8 - mark of the sixth student
mark[6] = 11; // 11 - mark of the seventh student
mark[7] = 12 // 12 - mark of the eighth student
mark[8] = 11; // 11 - mark of the ninth student
mark[9] = 6; // 6 - mark of the tenth student
mark[10] = 8; // 8 - mark of the eleventh student
mark[11] = 10; // 10 - mark of the twelfth student
```

Далее нам необходимо запросить номер студента у пользователя, чтобы узнать его оценку. Поэтому создаем целочисленную переменную.

```
int number = 0;
cout << "Enter a number from the students list: " << endl;
cin >> number;

// print a mark of the selected student
cout << "Mark is " << mark[number-1] << endl;
```

Заметим, чтобы вывести оценку первого студента нужно будет обращаться к нулевой ячейке, а для второго студента к ячейке с индексом один и так далее. Поэтому в выводе использовалась запись **mark[number-1]**.

Кроме этого, обратите внимание, что индекс послед него элемента никогда не будет равен числу размерности массива. Потому что индексация начинается с нуля, а не с единицы. Давайте теперь попытаемся разобраться, по чему это именно так.

Расположение массива в памяти

Память представляет собой массив байт, каждый из которого имеет свой уникальный адрес.

Имя массива как раз и хранит в себе адрес, что указывает на место его размещения. Первым в памяти находится нулевой, потом первый и т.д. Элементы располагаются по возрастанию адреса. Один элемент массива отстоит от другого на количество байт, равное базовому типу массива (рис. 4). Формула, по которой производится позиционирование по массиву:

```
базовый адрес + размер базового типа * индекс;
```

Если указывается неправильный адрес, производится позиционирование базового адреса на адрес, вычисленный по формулам. При этом программа получает полный доступ к содержимому ячейки памяти, которая ей по сути не принадлежит. В результате этого может произойти ошибка на этапе выполнения.

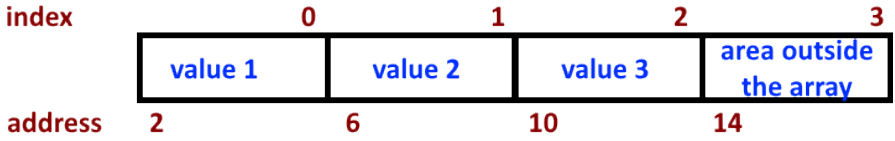

Рисунок 4

В заключение, следует отметить, что каждый элемента массива имеет свою собственную размерность, которая напрямую зависит от типа всего массива. Например, если массив имеет тип данных **int** — размер каждого элемента в нем — 4 байта. Таким образом, общий размер всего массива вычисляется по формуле:

```
ОБЩИЙ_РАЗМЕР = РАЗМЕР_ТИПА_ДАННЫХ*
  КОЛИЧЕСТВО_ЭЛЕМЕНТОВ_В_МАССИВЕ
```

Рассмотрим, как наш массив хранится в памяти:

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

int main() {
   // create a digit value array of three elements
   int train[3];

   // fill the array
   train[0] = 3;
   train[1] = 1;
   train[2] = 4;

   // output the array address
   cout << "Start address: " << train << endl;
   cout << "Array values: " << endl;
   cout << "train[0] =  " << train[0] << endl;
   cout << "train[1] =  " << train[1] << endl;
   cout << "train[2] =  " << train[2] << endl;
   cout << endl;

   return 0;

}
```

При этом поставим точку остановки на предпоследнюю строку **cout<<endl**. После этого запустим наш пример и обратимся в окно с переменными (рис. 5).

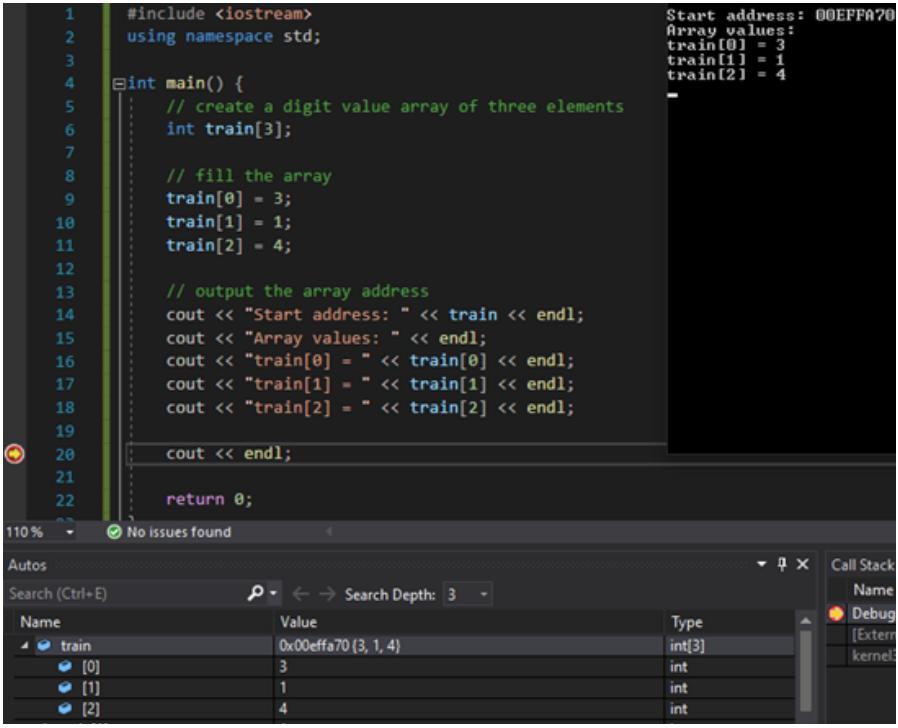

Рисунок 5

Как видим, имя массива хранит в себе адрес, где в памяти он располагается. В меню **Debug** (*Отладка*) выберем пункт **Windows** (*окна*), далее **Memory** (*память*), а после **Memory1** (*Память1*) (рис. 6).

В открывшемся окне прописываем нужный адрес и переходим на него (рис. 7).

Можем убедиться, что массив располагается в памяти последовательно, элемент за элементом.

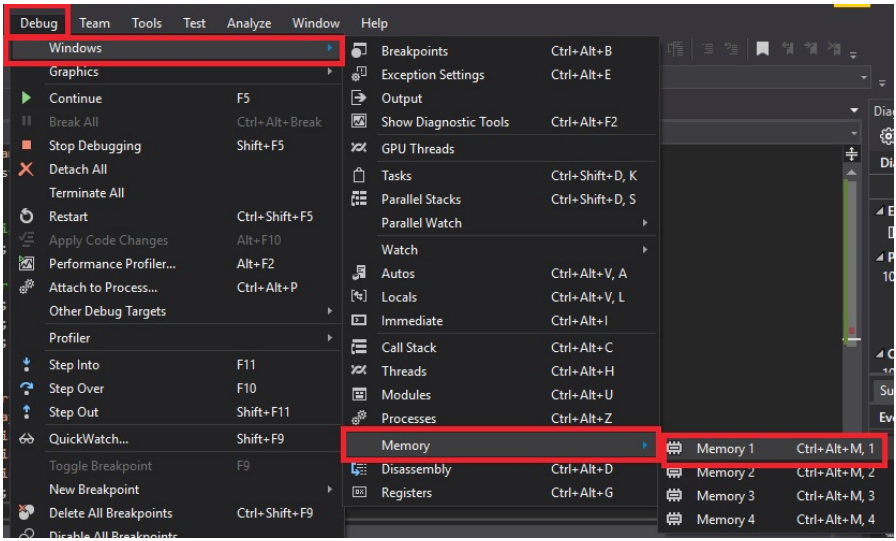

Рисунок 6

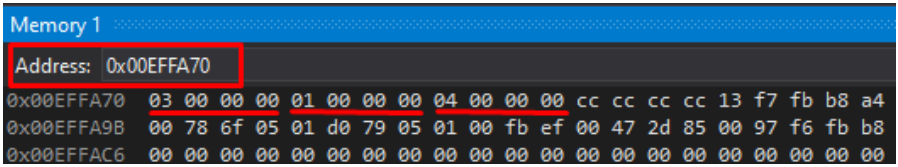

Рисунок 7

Теперь теоретически мы знаем о массиве почти всё. Осталось познакомиться с практической частью и убедиться, как легко и удобно создается и используется данная конструкция. Для этого переходим к следующему разделу урока.

2. Создание массива и заполнение его данными

Вариант первый

Объявлен массив **ar**, состоящий из 5 элементов, каждый из которых имеет тип данных **int**.

```
int ar[5];
```

Вариант второй

Объявлена константа **size**, значение которой равно **3**, а затем, массив **br**, состоящий из 3 элементов, каждый из которых имеет тип данных **double**.

```
const int size=3;
double br[size];
```

Примечание. *Мы рекомендуем вам использовать вторую форму записи, так как она является более корректной и удобной.*

Обращение к элементам массива

Рассмотрим, как обратиться к конкретному элементу массива.

```
// запись значения
имя_массива[индекс_элемента]=значение;

// получение значения
cout<<имя_массива[индекс_элемента];
```

Здесь, на место индекса_элемента может быть подставлено ЛЮБОЕ целочисленное значение, в том числе выражение, результатом которого является целое число.

Примечание. *Еще раз напоминаем — нумерация элементов в массиве начинается с нуля! Таким образом, в массиве из 5 элементов — последний элемент имеет индекс 4. Выходить за пределы массива нельзя, это приведет к ошибке на этапе выполнения.*

Варианты инициализации массива

Заполнить массив данными можно двумя способами:
Первый способ — инициализация при создании.

```
тип_данных имя_массива[количество элементов]=
{значение1, значение2, … значение n};
```

```
const int size=3;
int ar[size]={1,30,2};
```

При такой форме инициализации есть некоторые особенности:

1. Все значения списка инициализации имеют такой же тип данных, как и сам массив, поэтому при создании количество элементов можно не указывать.
   Операционная система сама определит размер массива исходя из числа элементов в списке инициализации.

```
тип_данных имя_массива[]={значение1, значение2,
                          значение3, … значение n};
```

```
int ar[]={1,30,2};

/*
    В данной строке массив автоматически получит
    размер 3.
*/
```

2. Если число элементов в списке инициализации меньше чем число элементов массива, то оставшиеся значения автоматически заполняются нулями:

```
int ar[5]={1,2,3}
```

такая запись эквивалентна записи:

```
int ar[5]={1,2,3,0,0};
```

3. Если значений в списке инициализации больше чем количество элементов массива, то происходит ошибка на этапе компиляции:

```
int array[2]={1,2,3}; // ошибка на этапе компиляции
```

4. При инициализации массивов можно использовать уже известную вам унифицированную инициализацию

```
int arr[]{1, 2, 3};
int arr2[4]{11, 21, 31};
```

Второй способ — инициализация массива при помощи цикла.

В этом случае заполнить массив значениями, можно с помощью пользователя.

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

int main()
{
    const int size = 3;
    int ar[size];// создание массива из трех элементов
                 // цикл перебирающий элементы массива
    for (int i = 0; i<size; i++)
    {
        cout << "Enter element\n";
        /* на каждой итерации цикла пользователю
           подставляется элемент с индексом i для
           заполнения. Секрет в том, что i - каждый
           раз новое значение
        */
        cin >> ar[i];
    }
    return 0;
}
```

Показ содержимого массива на экран

Вы, наверняка уже догадываетесь, что большинство операций с массивами удобнее проводить с помощью циклов, по очереди перебирая элементы. Это действительно так и показ на экран не является исключением. Приведем пример полной программы, которая создает, заполняет и показывает на экран массив.

```
#include <iostream>
using namespace std;
int main()
{
```

```
    const int size = 3;
    int ar[size]; // создание массива из трех элементов
                  // цикл, перебирающий элементы
                  // массива
    for (int i=0;i<size;i++)
    {
        cout << "Enter element\n";
        /* на каждой итерации цикла пользователю
           подставляется элемент с индексом i для
           заполнения. Секрет в том, что i - каждый
           раз новое значение
        */
        cin >> ar[i];
    }

    cout << "\n\n";
    // цикл, перебирающий элементы массива
    for (int i = 0; i<size; i++)
    {
        // показ элемента с индексом i на экран
        cout<<ar[i]<<"\n";
    }
    return 0;
}
```

Теперь, когда мы с вами познакомились с массивами, давайте перейдем к следующим разделам урока рассмотрим несколько практических примеров работы с ними.

3. Примеры программ работы с массивом

Задача 1

Написать программу, которая находит сумму всех отрицательных значений в массиве.

Код реализации

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

int main()
{
    // определение размера массива
    const int size=5;
    // создание и инициализация массива данными
    int ar[size]={23,-11,9,-18,-25};
    // переменная для накопления суммы
    int sum = 0;
    // цикл, перебирающий по порядку элементы массива
    for (int i=0;i<size;i++)
    {
        // если значение элемента отрицательное
        // (меньше нуля)
        if (ar[i]<0)
            sum += ar[i]; // добавить его значение
                          // к общей сумме
    }
    // показ значения суммы на экран
    cout<<"Sum = "<<sum<<"\n\n";
    return 0;
}
```

Комментарий к коду

1. Цикл поочередно перебирает элементы от **0** до **size**. При этом **size** не входит в проверяемый диапазон, т.к. индекс последнего элемента **size–1**.
2. На каждой итерации цикла происходит проверка содержимого элемента на отрицательное значение.
3. Если значение меньше нуля, оно прибавляется к сумме.

Как видите, работа с массивом очень похожа на анализ какого-то диапазона. Только, в данном случае, минимальная граница диапазона — **0**, а максимальная — определяется количеством элементов в массиве.

Задача 1

Написать программу, которая находит минимальное и максимальное значение в массиве и показывает их на экран.

Код реализации

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

int main()
{
    // определения количества элементов массива
    const int size=5;
    // создание и инициализация массива
    int ar[size] = { 23,11,9,18,25 };
    int max = ar[0]; // пусть 0 элемент максимальный
    int min=ar[0];   // пусть 0 элемент минимальный
                     // цикл перебирает элементы
                     // массива, начиная с 1-ы
```

```
    for (int i=1;i<size;i++)
    {
        // если текущий элемент меньше, чем минимум
        if (min>ar[i])
            // перезаписать значение минимума
            min = ar[i];
        // если текущий элемент больше, чем максимум
        if (max<ar[i])
            // перезаписать значение максимума
            max=ar[i];
    }

    // вывод результата на экран
    cout<<"Max = "<<max<<"\n\n";
    cout<<"Min = "<<min<<"\n\n";
    return 0;
}
```

Комментарий к коду

1. Для начала, выдвигаем предположение, что минимальным является элемент массива с индексом **0**.
2. Записываем значение элемента с индексом **0** в переменную **min**.
3. Затем, для того, чтобы либо подтвердить, либо опровергнуть этот факт, перебираем все элементы массива, начиная с элемента с индексом **1** в цикле.
4. На каждой итерации цикла, сравниваем предполагаемый минимум с текущим элементом массива (элемент с индексом **i**).
5. Если встречается значение меньше, чем предполагаемый минимум — значение **min** перезаписывается на меньшее найденное значение и анализ продолжается.

Все вышеописанные действия справедливы и для максимума, только осуществлять необходимо поиск большего значения. Теперь, когда вы знакомы с массивами и рассмотрели несколько примеров, пора сделать что-то самим. Желаем удачи в прохождении теста и выполнении домашнего задания.

4. Домашнее задание

1. Дана программа, которая определяет последнее положительное и первое отрицательное число в массиве. Найти и исправить синтаксические и логические ошибки.

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

int main()
{
    double size = 8;
    double arr[size]={-5.7, 6.0, 2, 0, -4.7, 6,
                      8.1, -4, 0};
    int positive = 0;
    for (int i = size; i >= 0; i++)
    {
        if (arr[i] > 0)
        {
            positive = arr[i];
            break;
        }
    }

    int negative = 0;
    for (int i = 0; i < size; i++)
    {
        if (arr[i] < 0)
        {
            negative = arr[i];
            break;
        }
    }
```

```
    cout << "Last positive number: " << positive <<
            endl;
    cout << "First negatine number: " << negative <<
            endl;
    return 0;
}
```

Входными данными во всех описанных ниже заданиях является массив из 10 элементов, заполненный пользователем с клавиатуры.

2. В массиве хранится информация о количестве жильцов каждой квартиры пятиэтажного дома (4 подъезда, на каждом этаже по 2 квартиры).
   а) по выбранному номеру квартиры определить количество жильцов, а также их соседей проживающих на одном этаже;
   б) определить суммарное количество жильцов для каждого подъезда;
   в) определить номера квартир, где живут многодетные семьи. Условно будем считать таковыми, у которых количество членов семьи превышает пять человек.
3. Дана температура воздуха за каждый день января. Определить:
   а) среднюю температуру за месяц;
   б) сколько раз температура воздуха опускалась нижеуказанной метки.
4. В массиве хранится информация о стоимости 10 марок автомобилей. Определить сумму наиболее дорогого автомобиля и узнать его номер. Если таких автомобилей несколько, определить:

а) номер первого такого автомобиля;
б) номер последнего такого автомобиля.

5. Написать программу, которая находит в массиве значения, повторяющиеся два и более раз, и показывает их на экран.

6. Заполнить два целочисленных массива **A[10]** и **B[10]**. Сформировать третий массив **X[20]**, элементы которого будут взяты из **A** и **B** в порядке:
а) чередования **(a0, b0, a1, b1, a2, b2, …, a9, b9)**;
б) следования **(a0, a1, a2,…, a9, b0, b1, b2,…, b9)**.