Оглавление

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. А, дело все в том, что каждая из переменных, рассмотренных в предыдущих уроках способна одновременно хранить лишь один элемент информации. Чтобы сохранить второй, необходимо создать еще одну переменную. Но что делать, если нужно хранить множество элементов однородных типов данных. 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.

1. Массив – это совокупность переменных, которая позволяет хранить несколько однотипных значений.

2. Все значения этой совокупности объединены под одним именем.

3. При этом каждая переменная в массиве является самостоятельной единицей под названием - *элемент*.

4. Каждый элемент имеет свой порядковый номер - *индекс*. По индексу можно обращаться к конкретному элементу массива.

5. Нумерация элементов в массиве начинается с нуля.

Схема:

Исходя из вышеописанных утверждений - общая схема представления массива будет выглядеть так:

Расположение массива в памяти:

Массив располагается в памяти последовательно, элемент за элементом. Сначала лежит нулевой, потом первый и т.д. Элементы располагаются по возрастанию адреса: Один элемент массива отстоит от другого на количество байт, равное базовому типу массива. 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.

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

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.

Создание массива и заполнение его данными.

Синтаксис объявления массива.

Для начала, нам необходимо научиться создавать массив. А, для этого - во-первых, определить общий синтаксис. Во-вторых, выяснить, каким правилам и ограничениям этот синтаксис подчиняется.

```
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 - **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.

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.

```
const int size=5;
int ar[size]; // создание массива
ar[2]=25; // запись значения 25 в элемент с индексом 2
// вывод на экран значения элемента с индексом 2 - 25
cout<<ar[2]<<"\n";
```

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
```

Второй способ - инициализация массива при помощи цикла.

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

```
#include<iostream>
using namespace std;
void 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];
   }
}
```

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;
```

```
void 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 (i=0;i<size;i++)
   {
      //показ элемента с индексом i на экран
      cout<<ar[i]<<"\n";
   }
}
```

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

Пример программы для нахождения суммы отрицательных элементов массива.

Постановка задачи:

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

Код реализации:

```
#include <iostream>
```

```
using namespace std;
void 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";
}
```

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.

Пример программы для нахождения минимального и максимального элементов массива.

Постановка задачи:

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

Код реализации:

```
#include <iostream>
```

```
using namespace std;
void 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";
}
```

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, что бы 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.

Домашнее задание

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

1. Написать программу, которая выводит содержимое массива наоборот.

```
Пример: массив 23 11 6 превращается в 6 23 11.
```

2. Написать программу, которая находит сумму четных и сумму нечетных элементов массива.

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

4. Написать программу, которая находит в массиве самое маленькое нечетное число и показывает его на экран.