# МАССИВЫ

6 лекция

Дисциплина «Программирование микроконтроллеров на языке С++»

# Понятие массивов

- Массивы являются важнейшим средством языка, применяемым во многих программах. Их использование позволяет удобным образом разместить в памяти большое количество необходимой информации. В массивы объединены результаты экспериментов, списки фамилий и т.п.
- **Массив — именованный набор с фиксированным количеством однотипных данных** (double, float, int ..)
- В C++ массив, как и любая переменная, должен быть объявлен. Из объявления массива компилятор должен получить информацию о типе элементов массива и их количестве.

# Объявление массивов

Объявление массива имеет формат:

**<Тип> <Имя_массива> [<размер массива>];**

- Тип задает тип элементов хранящихся в массиве.
- Размер массива – константное положительное целочисленное выражение, определяющее количество элементов массива.

```
int a[6] ;
```

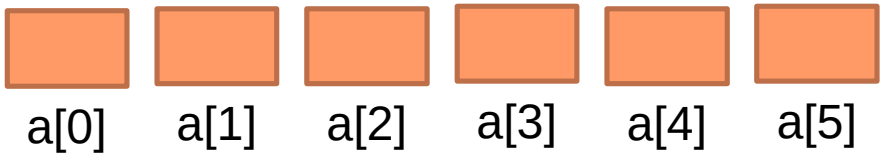
- Число элементов в массиве (размер) может быть опущено если  при объявлении массив инициализируется:

```
int a[]={0,1,2,3,4,5};
```

# Объявление массивов

- В современных языках программирования (PHP, C++, C#) нумерацию элементов массива принято начинать с нуля.

int a[6] ;




- Элементы массива размещаются в памяти последовательно.
- Доступ к элементу массива осуществляется по его номеру (индексу).
- Индекс (порядковый номер) – числа, определяющие место элемента в массива.

# Одномерные массивы

**Одномерный массив** – это структурированный набор однотипных элементов, к каждому из которых можно обратиться по его индексу, представляющему собой целое число, определяющее позицию элемента в массиве.

Пример объявление одномерного целочисленного массива: **int a[20];**

Здесь **a** – имя массива, состоящего из двадцати элементов. Первый элемент массива с индексом **0** записывается **a[0]**; второй с индексом **1– a[1]** и т.д. до двадцатого элемента с индексом **a[19]**

Отдельный элемент массива указывается по его имени и индексу в квадратных скобках.

# Обращение к отдельным элементам массива

Чтобы получить доступ к отдельным элементам массива, после имени массива мы пишем инструкцию, называемую индексной, заключённую в квадратные скобки. Эта инструкция определяет, к какому компоненту осуществляется доступ. Результат индексной инструкции должен быть целым числом. Простейшей формой индексной инструкции является константа.

***Пример.*** Объявляется массив, состоящий из трех элементов типа float

```
float array[3];
```

Для массива array, последовательность выражений присваивания

```
array[0]=5.2;
array[1]=-23.7;
array[2]=0.8;
```

заполнит все компоненты соответствующими значениями.

# Обращение к отдельным элементам массива

С каждым элементом массива можно работать точно так же, как и с простой переменной:

```
array[2]=9.3;          //присвоить новое значение
cin >> array[2];       //считать в него значение
cout << array[2];  //вывести содержимое элемента на экран
y=sqrt(array[2]);      //передать его в качестве параметра
x=3.14*angle[2]+9.3;   //использование в выражении.
```

# Инициализация массивов

Массивы также можно инициализировать при объявлении, как и переменные, однако при этом будет другой синтаксис. В случае инициализации массива при объявлении требуется указать начальные значения элементов массива, разделяя их запятыми и заключая весь список в фигурные скобки:

```
int array[4]={23, 45, 12, 8};
```

В этом объявлении элемент **array[0]** инициализирует значением **23**, элемент **array[1]** – значением **45**, ….

В фигурных скобках должно быть по крайней мере одно значение. Если в скобках значений больше, чем есть элементов, то произойдёт синтаксическая ошибка. Если значений окажется меньше, то оставшиеся элементы будут приравнены к нулю.

# Инициализация массивов

Особенностью языка C++ является то, что он позволяет опустить размер массива, если вы инициализируете его при объявлении:

```
float array[]={23.7, 45.1, 12.6, 0.8};
```

Компьютер вычисляет размер массива (в данном случае – 4) в соответствии с количеством перечисленных начальных значений.

# Основные операции над массивами

- Ввод элементов массива с консоли
- Печать элементов на экран
- Поиск минимального (максимального) элемента
- Задание элементам массива случайных значений
- Сортировка элементов массива.

# Ввод элементов массива с консоли

```
const int MAX_SIZE = 20;

int mas[MAX_SIZE];
int n;
cin >> n;
for (int i = 0; i < n; i++)
 cin >> mas[i];
```

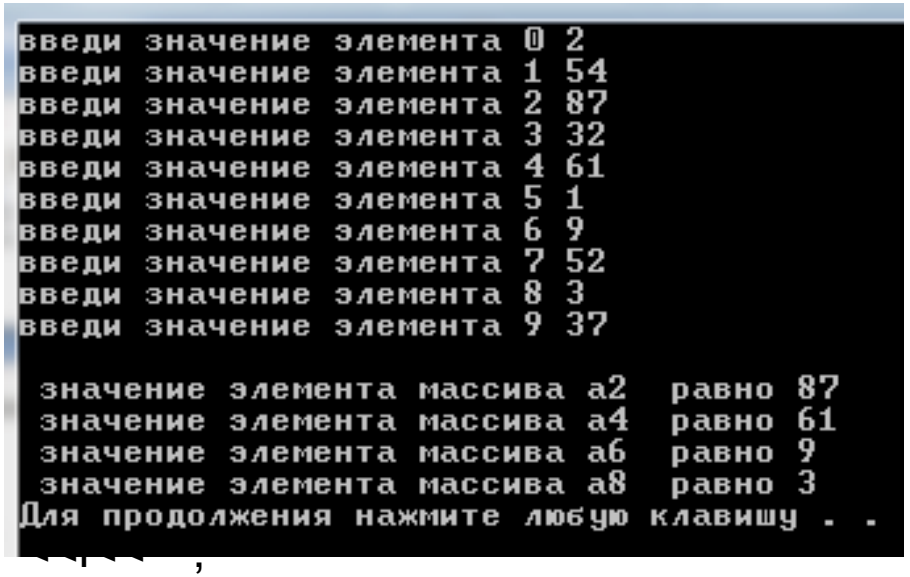
# Печать элементов массива на экран

```
int mas[100];
int n; //Количество элементов в массиве
……
cout << “mas : “ << endl;
for (int i = 0; i < n; i++)
  cout << mas[i] << “ “;
cout << endl;
```

# Одномерные массивы

Пример. Ввести одномерный массив из 10 целых чисел. Вывести четные по номеру индекса элементы этого массива.

```
#include <iostream>
#include <clocale>
using namespace std;
        int main()
        {
setlocale(LC_CTYPE, "russian");
        int i, a[10];
        for (i=0; i<10; i++)
        {
cout<<"введи значение элемента
cin >> a[i];
        }
        for (i=2;i<10;i+=2)
        {
        cout<<" \n значение элемента массива a"<<i<<"  равно "<< a[i];
        }
system ("pause");
return 0;
        }
```



# Поиск минимального (максимального) элемента в массив

```
int min = mas[0];
for (int i = 1; i < n; i++)
{
  if (mas[i] < min)
  {
      min = mas[i];
  }
}
cout << "min = " << min << endl;
```

- Нахождение максимального элемента в массиве осуществляется аналогично

# Сортировка массива

- Сортировка обычно выполняется путём повторного сравнения пар элементов массива и замены значений, если они отвечают определённым критериям.
- Порядок, в котором эти элементы сравниваются, зависит от того, какой алгоритм сортировки используется.
- Критерии состоят из того, как будет сортироваться массив (например, в порядке возрастания или в порядке убывания).

# Виды сортировок массивов

- Сортировка пузырьком / Bubble sort
- Сортировка выбором / Selection sort
- Быстрая сортировка / Quicksort
- Шейкерная сортировка / Shaker sort
- Сортировка расческой / Comb sort
- Сортировка вставками / Insertion sort
- Сортировка Шелла / Shellsort
- Сортировка деревом / Tree sort
- Пирамидальная сортировка / Heapsort
- Сортировка слиянием / Merge sort
- и т.д.

# Пузырьковая сортировка (Bubble sort)

- **Алгоритм пузырьковой сортировки** — это довольно простой в реализации алгоритм для сортировки массивов. Назван так, потому что большее или меньшее значение «всплывает» (сдвигается) к краю массива после каждой итерации, это будет видно в примере.

# Пузырьковая сортировка (Bubble sort)

- При пузырьковой сортировке сравниваются соседние элементы и меняются местами, если следующий элемент меньше предыдущего. Требуется несколько проходов по данным. Во время первого прохода сравниваются первые два элемента в массиве. Если они не в порядке, они меняются местами и затем сравнивается элементы в следующей паре. При том же условии они так же меняются местами. Таким образом сортировка происходит в каждом цикле пока не будет достигнут конец массива.

# Пузырьковая сортировка (Bubble sort)

- Алгоритм работает медленно с большими массивами, а поэтому малоэффективен и **на практике используется редко**, чаще всего в учебных целях. Для сортировки массивов на практике используют другие более быстрые алгоритмы, один из них — QuickSort (быстрая сортировка). Функция для быстрой сортировки включена во многие стандартные библиотеки языков программирования, например в языке C функция **qsort()** из стандартной библиотеки.

# Сортировка выбором (Selection sort)

- Для того, чтобы отсортировать массив в порядке возрастания, следует на каждой итерации найти элемент с наибольшим значением. С ним нужно поменять местами последний элемент. Следующий элемент с наибольшим значением становится уже на предпоследнее место. Так должно происходить, пока элементы, находящиеся на первых местах в массиве, не окажутся в надлежащем порядке.

```
#include <iostream>
#include <clocale>
#include <cstdlib>
using namespace std;
#define N 5
int main()
{
setlocale(LC_CTYPE,"russian");
cout <<"Сгенерировать массив и отсортировать его по возрастанию" <<endl;
int masiv[N];
cout <<"Сгенерированный массив" <<endl;
        for (int i=0; i<N; i++)
        {
        masiv[i]= rand()%100;
        cout <<masiv[i] <<endl;
        }
int min,x,k,j;
cout <<"Отсортированный массив" <<endl;
        for (int i=0; i<N; i++)
        {
        min=masiv[i];
        k=i;
                for(j=i;j<N;j++)
                                if(min>masiv[j])
                                {
                                min=masiv[j];
                                k=j;
                                };
                x=masiv[i];
                masiv[i]=masiv[k];
                masiv[k]=x;
        cout <<masiv[i] <<endl;
        }
system ("pause");
return 0;
}
```

```
Сгенерировать массив и отсортировать его по возрастанию
Сгенерированный массив
41
67
34
0
69
Отсортированный массив
0
34
41
67
69
Для продолжения нажмите любую клавишу . . .
```

## Сортировка массива методом простого выбора

В некоторых случаях возникает необходимость заполнить массив произвольными значениями чисел. Для этого применяются функции rand, srand

```
#include <cstdlib>   // случайные числа
```

**RAND_MAX** – максимальное случайное целое число (обычно RAND_MAX = 32767)

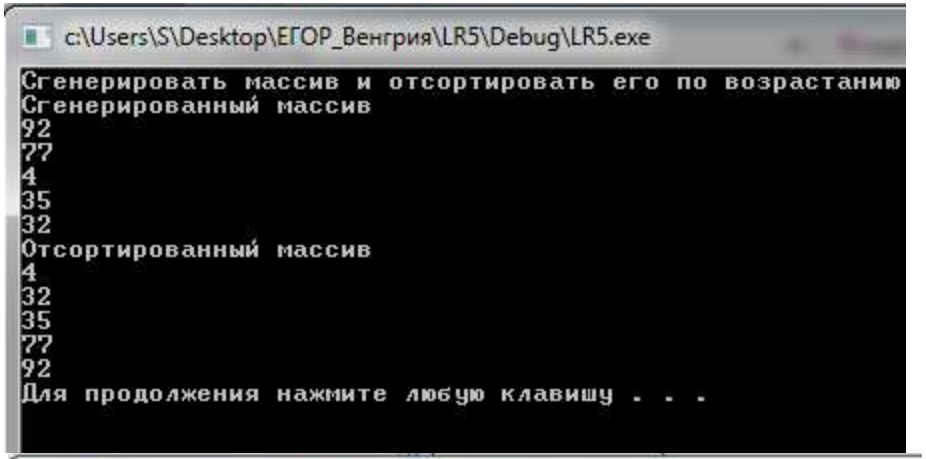
**rand()** – случайное целое число в интервале [0,RAND_MAX]

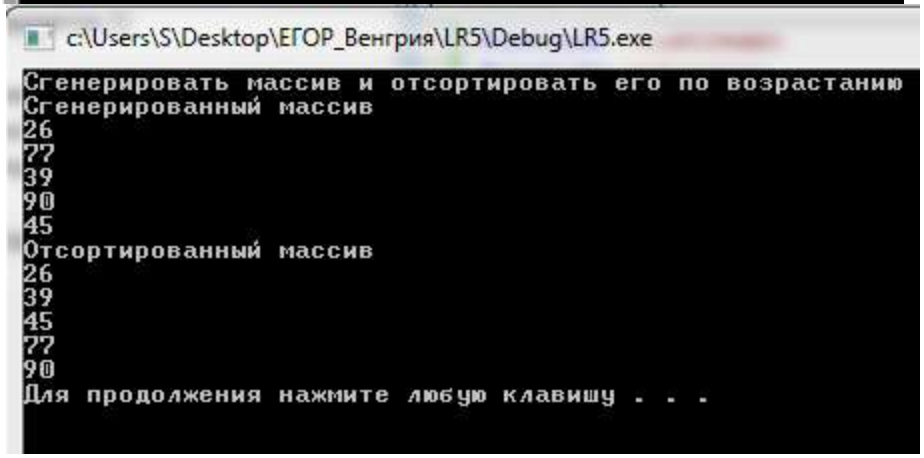
```
int x, y;
x = rand(); // первое число [0,RAND_MAX]
y = rand(); // уже другое число
```

**srand(N)** – установить начальное значение последовательности случайных чисел N:

```
srand ( 345 ); // начнем с 345
```

```
#include <iostream>
#include <clocale>
#include <cstdlib>
#include <ctime>
using namespace std;
#define N 5
int main()
{
setlocale(LC_CTYPE,"russian");
srand (time(NULL));
cout <<"Сгенерировать массив и отсортировать его по возрастанию" <<endl;
int masiv [N];
cout <<"Сгенерированный массив" <<endl;
        for (int i=0; i<N; i++)
        {
        masiv[i]= rand()%100;
        cout <<masiv[i] <<endl;
        }
int min,x,k,j;
cout <<"Отсортированный массив" <<endl;
        for (int i=0; i<N; i++)
        { min=masiv[i];
        k=i;
                for(j=i;j<N;j++)
                                if(min>masiv[j])
                                { min=masiv[j];
                                k=j;};
                x=masiv[i];
                masiv[i]=masiv[k];
                masiv[k]=x;
        cout <<masiv[i] <<endl;
        }
system ("pause");
return 0;
}
```



# Встроенные функции для сортировки

- Остановимся на встроенных в C++ функциях qsort и std::sort. Важно заметить, что можно реализовать и использовать и другие самые разнообразные функции для сортировки массива.
- Функция qsort библиотеки stdlib.h:

**qsort (<сортируемый массив>, <количество элементов>, sizeof(<тип элемента>), <функция для сравнения>);**

- Алгоритм работает очень просто. Программа проходит по всем элементами массива по порядку. Каждый текущий элемент сравнивается с соседним и, если он меньше/больше(если сортируем по убыванию/возрастанию соответственно) меняется местами с соседним.

# Встроенные функции для сортировки

- Функция для сравнения - наиболее своеобразная часть кода.  Тип данный int для массива другого типа можно заменить на другой.

```
#include <stdlib.h>     // подключаем qsort
#include <iostream>     // подключаем cout

//функция для сортировки по возрастанию
int comp1 (const void * a, const void * b)
{
  return ( *(int*)a - *(int*)b );
}

//функция для сортировки по убыванию
int comp2 (const void * a, const void * b)
{
  return ( *(int*)b - *(int*)a );
}

using namespace std; // чтобы не подключать std
```

# Встроенные функции для сортировки

- Полный пример её использования выглядит так:

```
int main ()
{
  int sort_int_array[] = { 13, 12, 43, 91, 30, 15, 74, 14 };
  int i;//счётчик

  //сортировка по возрастанию
  qsort (sort_int_array, 8, sizeof(int), comp1); //12, 13, 14, 15, 3
0, 43, 74, 91

  //ввывод результатов на экран
  for(i=0; i<8; i++)
    cout<<sort_int_array[i]<<" ";
  cout<<"\n";

  //сортировка по убыванию
  qsort (sort_int_array, 8, sizeof(int), comp2);//91, 74, 43, 30, 1
5, 14, 13, 12

  //ввывод результатов на экран
  for(i=0; i<8; i++)
    cout<<sort_int_array[i]<<" ";
  cout<<"\n";

  system("pause");
}
```

# Многомерные массивы

Массивы в C++ могут иметь несколько индексов.

- **Двумерные массивы** представляют собой таблицы значений, содержащие данные в **строках и столбцах**.

  Первый индекс указывает **номер строки**, второй — **номер столбца**.
- В C++ можно объявлять массивы с несколькими измерениями. На практике используются в основном двумерные и трёхмерные массивы, так как увеличение числа измерений значительно усложняет работу с данными и требует больше памяти. Некоторые компиляторы позволяют создавать массивы с числом измерений до 12, однако это встречается крайне редко.

# Объявление многомерного массива

- Многомерные массивы задаются указанием нескольких размеров, каждый из которых заключен в квадратные скобки.
- Пример объявления двумерного массива:

```
double array[8][2];
```

Такое объявление задает таблицу из 8 строк и 2 столбцов.

# Инициализация многомерного массива

Многомерные массивы могут получать начальные значения точно также, как и одномерные массивы. Например, двумерный массив можно объявить и дать ему начальные значения следующим образом:

```
int a[2][3] = {{1,2,3},{4,5,6}};
```

Здесь:

- внешние фигурные скобки охватывают весь массив;
- внутренние пары скобок представляют строки таблицы.

Таким образом:

a[0] = {1, 2, 3} // первая строка

a[1] = {4, 5, 6} // вторая строка

# Заключение

- Массивы являются одним из базовых средств организации и обработки данных. Они позволяют хранить и обрабатывать набор однотипных элементов, упрощая доступ к ним по индексу.
- В задачах автоматизации и управления массивы широко используются для хранения и обработки измеренных параметров, сигналов датчиков и управляющих воздействий.
- Одномерные массивы применяются для линейных наборов данных, двумерные - для таблиц и матриц.
- Для обработки массивов применяются алгоритмы сортировки и анализа данных, что обеспечивает упорядочение информации и повышение надёжности алгоритмов управления.