**Лекция 6**

# ОДНОМЕРНЫЕ МАССИВЫ И ИХ ОБРАБОТКА В VISUAL C++

**Цель лекции:** изучить особенности обработки одномерных массивов в среде Visual C++

**Основные вопросы лекции.**

1. Одномерные массивы данных и их инициализация в среде Visual C++.
2. Консольный ввод и вывод одномерных массивов в среде Visual C++.
3. Присваивание и копирование одномерных массивов в Visual C++.
4. Поиск элемента в одномерных массивах в Visual C++.
5. Перестановка и сортировка элементов в одномерных массивах.

## 1. Одномерные массивы данных и их инициализация в Visual C++

**Массив** – это конечная именованная последовательность однотипных величин.

**Массив** в информатике – это некоторое множество мест в памяти компьютера, называемых **элементами массива**, к которым можно обратиться по одному имени переменной. Каждый из **элементов** хранит единицу данных определенного типа (тип данных одинаков для всех элементов массива).

Каждый элемент массива определяется именем массива и его порядковым номером в массиве (индексом). Индекс элемента массива – всегда целое число.

Массивы бывают одномерными и многомерными. В данной лекции исследуем работу с **одномерными массивами в Visual C++**. Одномерные массивы часто называют **векторами.**

Для использования массива в программе его необходимо **объявить**, т. е. зарезервировать под массив определенное количество ячеек памяти.

При объявлении массива указывается тип элементов массива, имя массива и его размер:

**Тип Имя массива[размер]; .**

Например, оператор

**int A[8];**

описывает целый одномерный массив по имени **A** из 8-и целых чисел. В памяти будет зарезервировано место для 8-и целочисленных элементов массива (таб. 1).

**Таблица 1.** Значения и расположение в памяти элементов одномерного массива A**[8]**

| Элемент массива | **A[0]** | **A[1]** | **A[2]** | **A[3]** | **A[4]** | **A[5]** | **A[6]** | **A[7]** |
|---|---|---|---|---|---|---|---|---|
| Индекс элемента, i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Значение элемента | 15 | 22 | 34 | 57 | 11 | 29 | 89 | 47 |

Обращение к элементам массива осуществляется по имени массива с указанием индекса (номера элемента массива) в квадратных скобках:

```
A[7]= 47;
x = A[3];
в = A[5];
```

Индексация элементов в массиве начинается с ноля. Например, если объявлен массив

**int B[100];** ,

то элементы массива будут иметь следующие индексы:

**B[0], B[1], … B[99]**.

Тогда оператор

**x=B[13];**

означает, что переменной **x** будет присвоено значение 14-го элемента массива B.

Массив, как и переменную, можно инициализировать при объявлении. Значения для последовательных элементов массива отделяются один от другого запятыми и помещаются в фигурные скобки. Например,

**int C[6]= {2, 4, 7, 11, 12, 13};** .

Если в списке инициализации значений элементов указано меньше, чем размер массива, то имеет место частичная инициализация. При таком объявлении в выражении инициализации после последнего значения для наглядности ставят запятую:

**int C[6]= {2, 4, 7,};** .

При этом элементам **C[0]**, **C[1]** и **C[2]** будут присвоены значения 2, 4 и 7, а оставшиеся элементы массива инициализации не получат.

Кроме этого, инициализация массивов возможна в процессе выполнения программы – путем записи данных в отведенные для массивов ячейки памяти.

При работе с массивами, в т. ч. одномерными, целесообразно использовать оператор цикла **for**, т.к. известен размер обрабатываемого массива (число элементов массива), т. е. число повторений цикла.

В языке С++ не проверяется выход индекса за пределы массива. Если массив **m[100]** целочисленный массив:

**int m[100];** ,

а в программе указано

**x=m[200];** ,

то сообщение об ошибке не будет, а переменной **x** будет присвоено произвольное значение.

При операциях с массивами Visual C++ все действия в программе выполняются над элементами массива (!), а не над массивом в целом. При этом индекс элемента может быть задан либо его значением, либо выражением:

**A[4], F[i+k+1];** .

Над массивами можно выполнять следующие действия:

1. Вводить массивы в память компьютера.
2. Выводить массивы на экран дисплея, на другое устройство или в файл.
3. Присваивать определенные значения элементам массивов.
4. Копировать массивы.
5. Переставлять элементы массивов.
6. Сортировать элементы массивов.

## 2. Консольный ввод и вывод одномерных массивов в среде Visual C++

Т. к. все действия необходимо выполнять над элементами массива, то для ввода массива в память компьютера необходимо организовать его поэлементный ввод посредством оператора цикла **for** (т. к. известен размер массива).

**Пример 1**. Исследовать способы консольного ввода и вывода массива **A** из **N** целых чисел. Необходимо ввести с клавиатуры массив **A** в память, определить его размер и вывести эту информацию на дисплей.

Блок-схема алгоритма решения такой задачи приведена на рис. 8.1.

Реализующая данный алгоритм программа имеет следующий вид:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
int const N=8;                          //инициализация размерности массива
int i;                                  //объявление параметра цикла i
int A[N];                               //объявление одномерного массива  A
cout<<endl<<"Vvedite massiv:"<<endl;
    for(i=0; i<N; i++)                  //цикл по i для ввода массива  A
        cin>>A[i];                      //ввод i-го элемента массива  A
cout<<endl;
    for (i=0; i<N; i++)                 //цикл по i для вывода A на экран
        cout<<"  A["<<i<<"]= "<<A[i];   //вывод i-го элемента массива  A
cout<<endl<<"size= "<<i<<endl;          //вывод размерности массива  A
getch();
return 0;
}
```

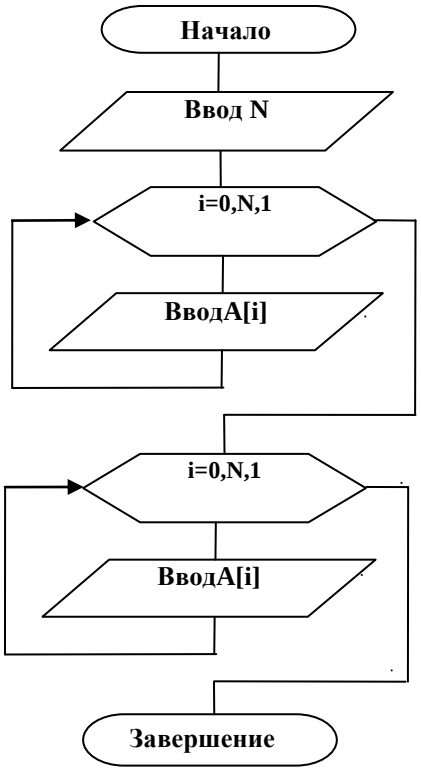


Рис. 1. Блок-схема алгоритма ввода и вывода элементов вектора с использованием цикла **for**

После запуска программы и ввода элементов массива **A** вид экрана будет следующий:

**Vvedite massiv:**
**1 2 3 4 5 6 7 8**

**A[0]= 1 A[1]= 2 A[2]= 3 A[3]= 4 A[4]= 5 A[5]= 6 A[6]= 7 A[7]= 8**
**size= 8**

### 3. Присваивание и копирование одномерных массивов в Visual C++

Элементам массива могут быть присвоены значения выражений. Элементы массива и значения выражений должны иметь один и тот же тип.

Например, объявлен массив **float A[3];**
тогда

```
A[0]= 3.5;
A[1]= 0;
A[2]= a*x + b;
```

**Копирование** – это присваивание значений элементов одного массива элементам другого массива. При копировании оба массива должны иметь одинаковый размер и тип элементов. Копирование массива **A** в массив **B** будет иметь вид:

```
for (i=0; i<N; i++) B[i]= A[i];
```

**Пример 2.** Задать массив **X** действительных чисел из 8 элементов непосредственным присваиванием в программном коде. Вычислить массив **Y** по заданной формуле:

$$Y_i = 2X_i + \sqrt[3]{X_i^5}\ .$$

Массив **Y** вывести на экран.

Блок-схема алгоритма решения этой задачи аналогична блок-схеме, приведенной на рис. 1.

Реализующая данный алгоритм программа имеет следующий вид:

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
int i;
```

```
double X[8]={1.2, 3.4, 12, 5.12, 23.1, 3, 0, 35.2};   //Инициализация элементов массива X[8]
double Y [8];
   for (i=0; i<8; i++)                                 //Цикл для вычисления и вывода элемента  Y[i]
   {                                                   //Начало составного оператора
       Y [i]=2*X[i]+pow(X[i], 5/3);                    //Вычисление элементов массива Y[8]
       cout<<"  X["<<i<<"]="<<X[i];                    //Вывод элементов массива X[8] на экран
       cout<<"  Y["<<i<<"]="<<Y[i];                    //Вывод элементов массива Y[8] на экран
       cout<<endl;                                     //Переход на новую строку
   }                                                   //Конец составного оператора
getch();
return 0;
}
```

В результате выполнения программы получим:

```
X[0]=1.2  Y[0]=3.6
X[1]=3.4  Y[1]=10.2
X[2]=12   Y[2]=36
X[3]=5.12Y[3]=15.36
X[4]=23.1Y[4]=69.3
X[5]=3    Y[5]=9
X[6]=0    Y[6]=0
X[7]=35.2Y[7]=105.6
```

### 4. Поиск элемента в одномерных массивах

Поиск элемента в массиве заключается в выделении из массива отдельных его элементов. Поиск может проводиться по образцу или по правилу.

**Поиск по образцу** заключается в следующем. Задается значение некоторой переменной (образец) и все элементы массива (или часть элементов) сравниваются со значением этой переменной (образцом).

**Поиск по правилу** проводится на основе проверки некоторых условий, которым должны отвечать либо элемент массива, либо группа элементов.

Рассмотрим пример поиска элемента по образцу.

**Пример 3.** Исследуем программу для поиска элемента одномерного массива по образцу. Задан массив **A** из 8 произвольных чисел. Необходимо определить количество элементов, которые больше заданного числа **q** и их порядковые номера. На экран вывести исходный массив, элементы, большие **q** и их порядковые номера.

**Решение**. Определим типы и структуры данных, которые будут использоваться в программе:
**q** – число, по которому осуществляется поиск элементов исходного массива **A;**
**i** – номер элемента исходного массива **A** (параметр цикла);
**A[8]** – массив из **N** произвольных чисел;
**X[8]** – массив для хранения номеров элементов исходного массива **A** , больших **q.**
**j** – номер элемента массива **X** для хранения номеров (параметр цикла);

Бл**о**к-схема алгоритма поиска элементов одномерного массива приведена на рис. 2**.**

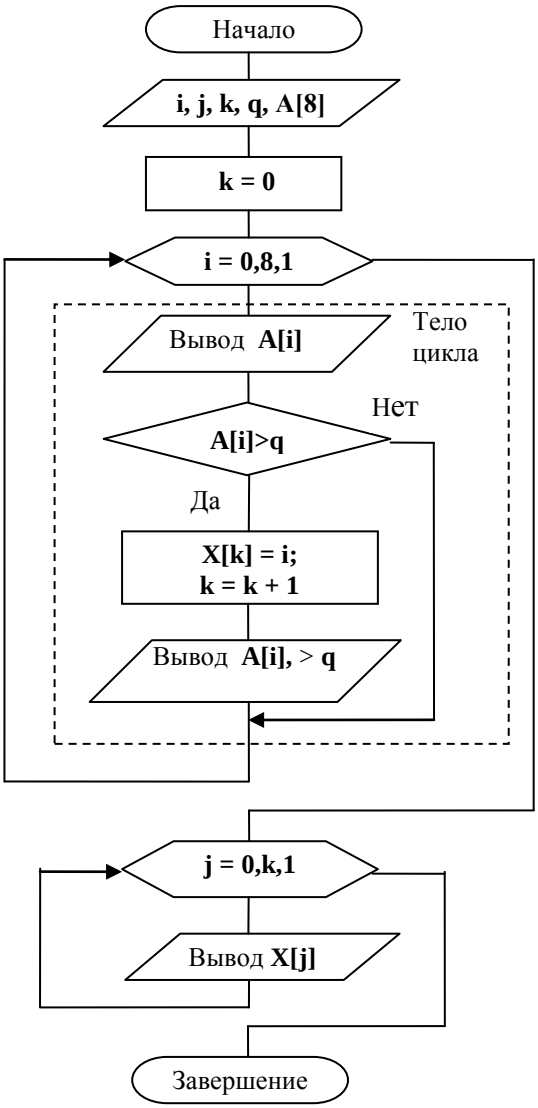


Рис. 8.2. Блок-схема алгоритма поиска элементов одномерного массива

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
int i, j, k;
double q=3.7;                          //Объявление перменной  q
double A[8]={1,  2,  3,  4,  5,  6,  7,  8};   //Инициализация элементов исходного массива A[8]
int X[8];                              //Объявление массива X[k] номеров элементов массива  A[8]
k=0;
   for (i=0; i<8; i++)                 // Цикл для перебора элементов массива A[8]
   {                                   //Начало составного оператора в цикле
      cout<<"  A["<<i<<"] = "<<A[i];   //Вывод всех элементов массива  A[8]
         if(A[i]>q)                    //Условный оператор для сравнения элементов массива A[8] с q
         {                             //Начало составного оператора в условном операторе
            X[k]=i;                    //Записываются номера элементов, больших q,  в массив X[k]
            k=k+1;
            cout<<"  >q:  A["<<i<<"] = "<<A[i]<<endl; //Вывод  элементов массива A[8],  > q
         }                             //Конец составного оператора в условном операторе
      else
      cout<<endl;
   }                                   //Конец составного оператора в цикле
cout<<endl<<"Nomera elementov  >q: ";
   for (j=0;j<k; j++)                  //Цикл для вывода номеров элементов массива A[8],  > q
      cout<<X[j]<<" ";                 //Вывод номеров элементов массива A[8],  > q, на экран
getch();
```

**return 0;**

В результате выполнения программы получим:

**A[0] = 1**
**A[1] = 2**
**A[2] = 3**
**A[3] = 4 >q: A[3] = 4**
**A[4] = 5 >q: A[4] = 5**
**A[5] = 6 >q: A[5] = 6**
**A[6] = 7 >q: A[6] = 7**
**A[7] = 8 >q: A[7] = 8**

**Nomera elementov >q: 3 4 5 6 7**

Исследуем программу для поиска элемента одномерного массива по правилу.

**Пример 4**. Задан массив **A** из 8 произвольных чисел. Найти максимальный элемент массива **A** и его номер. На экран вывести исходный массив **A**, максимальный элемент массива **A** и его номер.

**Решение**. Определим типы и структуры данных, которые будут использоваться в программе:
**i** – номер элемента исходного массива **A** (параметр цикла);
**A[8]** – массив из 8 произвольных чисел;
**B** – вспомогательная переменная для хранения максимального элемента массива для i-го шага;
**C** – вспомогательная переменная для хранения номера (индекса) максимального элемента массива.
Блок-схема алгоритма поиска максимального элемента одномерного массива приведена на рис. 3

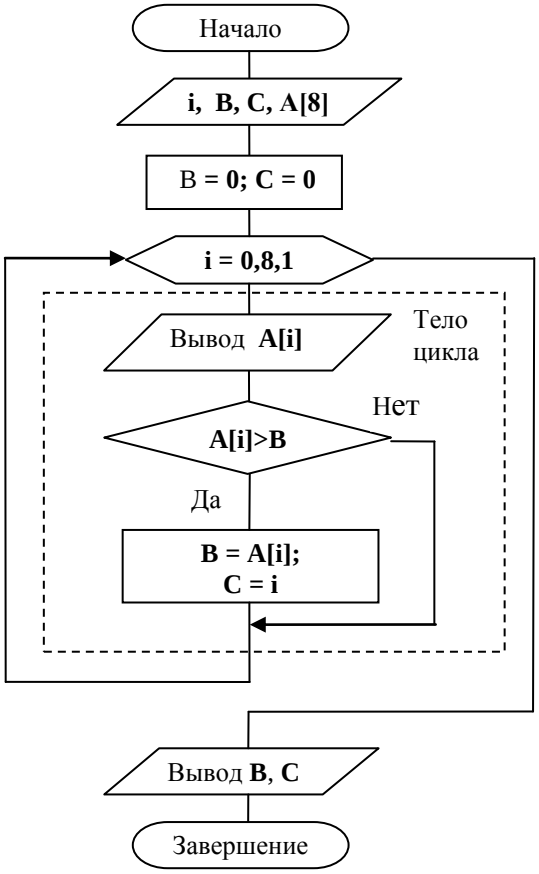


Рис. 3. Блок-схема алгоритма поиска максимального элемента одномерного массива

```
int _tmain(int argc, _TCHAR* argv[])
{
int i, C;                          //Объявление вспомогательной переменной C и индекса i
double B;                          //Объявление вспомогательной переменной B
double A[8]={1, 2, 3, 4, 5, 6, 7, 1};   //Инициализация элементов исходного массива A[8]
B=A[0];
C=0;
   for (i=0; i<8; i++)             //Цикл для перебора элементов массива A[8]
   {                               //Начало составного оператора в цикле
```

```
        cout<<"A["<<i<<"]="<<A[i];          //Вывод всех элементов массива  A[8]
        if(A[i]>B)                          //Условный оператор для сравнения элементов массива A[8]
        {                                   //Начало составного оператора в условном операторе
            B=A[i];                         //Присваивание  B значения максимального элемента
            C=i;                            //Присваивание переменной C номера максимального элемента
        }                                   //Конец составного оператора в условном операторе
    }                                       //Конец составного оператора в цикле
cout<<endl<<»Max element:  «<<B;
cout<<endl<<"N max elementa:   "<<C;
    getch();
return 0;
}
```

В результате выполнения программы получим:

```
A[0] = 1    A[1] = 2    A[2] = 3    A[3] = 4   A[4] = 5   A[5] = 6   A[6] = 7   A[7] = 1
Max element:  7
N max elementa:   6
```

### 5. Перестановка и сортировка элементов одномерных массивов в среде Visual C++

**Сортировка массива** – это упорядочивание элементов массива по определенным признакам. Например, необходимо упорядочить массив по возрастанию значений его элементов.

Эта задача решается путем многократного поиска максимального элемента в исходном массиве и пересылки его в новый упорядоченный массив.

Рассмотрим сортировку массива методом парной перестановки элементов массива.

Элементы массива **x[0], x[1], x[2], x[3]…x[n-1]** рассматриваются слева направо и по очереди сравниваются пары элементов:

**x[0]** и **x[1], x[1]** и **x[2], x[2]** и **x[3]…x[n-2]** и **x[n-1].**

Если **x[i-1] >x[i]**, то делается перестановка этих элементов. После того, как просмотрен весь массив, максимальный элемент будет расположен на последнем месте массива дело, то есть он будет иметь индекс **n-1**. При втором просмотре массива эта процедура повторяется только для массива **x[0], x[1], x[2], x[3]…x[n-2]**. При третьем просмотре – для массива **x[0], x[1], x[2], x[3]…x[n-3]**.

Рассмотрим способ парной перестановки соседних элементов одномерного массива. Эта задача решается с использованием вспомогательной переменной, играющей роль буфера.

Например, необходимо поменять местами элементы **A[i]** и **A[i+1]**. Введем дополнительную переменную **C** того же типа, что и элементы массива.

Алгоритм перестановки будет следующим (рис. 4).

1. Элемент **A[i]** помещаем в **C**
2. Элемент **A[i +1]** пересылаем на место **A[i]**.
3. Элемент **A[i]**, который находится в **C**, пересылаем на место **A[i+1]**.

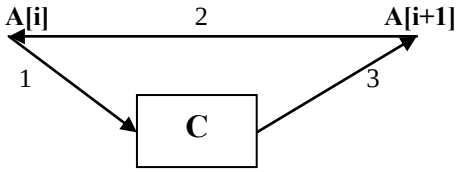


Рис. 4. Схема перестановки элементов одномерных массивов

Фрагмент программы перестановки двух элементов массива будет иметь следующий вид.

```
C = A[i];
A[i]= A[i+1];
A[i+1]= C;
```

**Пример 5**. Исследуем программу для сортировки одномерного массива по возрастанию.

Задан массив **A** из 15 произвольных чисел. Отсортировать элементы массива **A** по возрастанию. На экран вывести отсортированный массив **A**.

**Решение**. Определим идентификаторы и типы данных, которые будут использоваться в программе:

**i** – номер элемента исходного массива **A** (параметр цикла);

**A[15]** – исходный массив из 15 произвольных чисел (для удобства отладки программы составим массив **A** из чисел от 0 до14);

**B** – вспомогательная переменная для хранения максимального элемента массива для i-го шага;

```
#include "stdafx.h"
#include <conio.h>
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
int i,j;                                          //Объявление  индексов  i и j
double B;                                         //Объявление вспомогательной переменной B
double A[15]={13,2,0,4,12,6,10,14,9,7,11,5,1,8,3};  //Инициализация исходного массива A[15]
    for (j=0; j<15-1; j++)                        //Внешний цикл для перебора элементов массива A[15]
        for (i=0; i<15-1; i++)                    //Вложенный цикл для перебора элементов массива A[15]
                if(A[i]>A[i+1])                   //Условный оператор для сравнения элементов массива A[15]
                {
                    B=A[i];                       //Присваивание переменной B значения i-ого элемента
                    A[i]=A[i+1];                  //Перемена местами  i-ого и i+-ого элементов
                    A[i+1]=B;
                }
for (i=0; i<15; i++)                              //Цикл для вывода элементов отсортированного массива A[15]
    cout<<"  A["<<i<<"] = "<<A[i];
getch();
return 0;
}
```

В результате выполнения программы получим:

```
X[0] = 0  X[1] = 1  X[2] = 2  X[3] = 3  X[4] = 4  X[5] = 5  X[6] = 6  X[7] = 7
X[8] = 8  X[9] = 9  X[10] = 10  X[11] = 11  X[12] = 12  X[13] = 13  X[14] = 14
```

После отладки программы необходимо оставить только итоговый (на последнем шаге) вывод массива A.

Данная программа не является оптимальной. Видно, что исходный массив был отсортирован уже при **j**=10. Для устранения избыточных операций можно ввести некоторые критерии, но это усложнило бы понимание студентами рассматриваемых в данной работе алгоритмов.

**Выводы.** Под массивом в языке С++понимают набор данных одного и того же типа, собранных под одним именем. Отдельная единица таких данных, что входит в массив, называется элементом массива. В качестве элемента массива могут выступать данные любого типа (один тип данных для всех элементов массива). Каждый элемент массива определяется именем массива и его порядковым номером в массиве (индексом). Индекс элемента массива – всегда целое число.

При объявлении массива указывается тип элементов массива, имя массива и его размер. Объявление массива в программе – следующее.

**Тип Имя массива [размер];**

При обработке массива все действия в программе выполняются над элементами массива, а не над массивом в целом, как объектом. При обработке массива индекс элемента может быть задан либо его значением, либо выражением:

**A[4], F[i+k+1];**

Над массивом можно выполнять следующие действия:

1. Введение массива в память ПК.
2. Выведение массива на экран дисплея.
3. Присвоение определенных значений элементам массива.
4. Копирование массива.
5. Перестановка элементов массива.
6. Поиск элемента в массиве.
7. Сортировка массива.

**Вопросы для самоконтроля:**

1. Массив в информатике – это…

1) некоторое множество мест в памяти компьютера;

2) некоторое множество мест в памяти компьютера, называемых **элементами массива**, к которым можно обратиться по одному имени переменной;

3) некоторое множество мест в памяти компьютера, называемых **элементами массива**, к каждому из которых можно обратиться по своему имени.

2. Чем однозначно определяется каждый элемент массива?

1) Именем массива.

2) Именем массива и его порядковым номером в массиве (индексом).

3) Индексом элемента.

3. Какой индекс у 3-его элемента массива **int B[100]**?
   1) 3 2) 2 3) 0
4. Можно ли объявить массив следующим образом: **int C[6]= {2, 4, 7, 11, 12, 13}**?
   1) Да 2) Нет 3) В некоторых случаях
5. Как объявляется массив в языке С++?
   1) **int A[1..20]** 2) **mas: A[1..20] int** 3) **int A[20]**
6. Какое значение будет присвоено переменной **X** в **программе** на С++ оператором **X = m [13 ]**?
   1) 12-го элемента 2) 13-го элемента 3) 14-го элемента
7. Сколько ошибок сделано при записи оператора **if (A>7) B=A;** если обрабатывается массив **A**?
   1) 3 2) 2 3) 0
8. В программе на С++ при обработке массива все действия выполняются…
   1) над массивом в целом
   2) над элементами массива
   3) только над индексами элементов массива
9. Каким оператором можно ввести с клавиатуры **n** элементов массива **X**?
   1) **for (i=0; i<=n; i++) cin>>X[i]**
   2) **for (i=0; i< n; i++) cin>>X[i]**
   3) **for (i=1; i<=n; i++) cin>>X[i]**
10. Как в программе на С++ будет выведен на экран массив **A[k]**оператором **for(i=0; i<k; i++) cout<<A[i]<<endl;**?
   1) В виде строки 2) В виде столбца 3) В виде матрицы
11. Любой массив в языке С++…
   1) имеет имя 2) не имеет имени 3) имена имеют только элементы массива
12. Задан массив **X = {X1, X2 ..., XN}**. Каким оператором можно вывести этот массив на экран?
   1) **cout<<X;**
   2) **for (i=0; i<n; i++) cout<<X;**
   3) **for (i=0; i<n; i++) cout<<X[i];**
13. В программе на С++ объявлен массив **int B[10];** Это означает, что массив **B** содержит…
   1) 9 элементов 2) 10 элементов 3) 11 элементов
14. Возможна ли следующая инициализация массива **B** программе на С++: **int B[6 ]={2, 4, 7};**?
   1) Возможно 2) Не возможно 3) Зависит от мастерства программиста
15. В программе на С++ при обработке массива индекс элемента массива может быть задан…
   1) Значением 2) Выражением 3) Не важно как
16. Есть массивы **A** и **B** одинакового размера и их элементы имеют тип **int**. Возможно ли в языке С++ копирование массива **A** в **B** операцией **B = A**?
   1) Возможно 2) Невозможно 3) В некоторых случаях
17. При каком условии можно копировать массив **A** в массив **B**?
   1) Массивы **A** и **B** разного размера и их элементы имеют одинаковый тип
   2) Массивы **A** и **B** одинакового размера и их элементы имеют разный тип
   3) Массивы **A** и **B** одинакового размера и их элементы имеют одинаковый тип
18. Какой оператор копирует массив **A[N]** в массив **B[N]**?
   1) **for (i=0; i<N; i++) B[i]=A[i];**
   2) **for (i=0; i<K; i++) B[i]=A[i];**
   3) **for (i=0; i<N; i++) A[i]=B[i];**
19. Какой результат выполнения оператора **for (i=0; i<8; i++) cout<<"A["<<i<<"]="<<A[i];** (**A**=1…8)?
   1) **A[0]=1 A[1]=2 A[2]=3 A[3]=4 A[4]=5 A[5]=6 A[6]=7 A[7]=8 A[8]=8**
   2) **A[1]=1 A[2]=2 A[3]=3 A[4]=4 A[5]=5 A[6]=6 A[7]=7 A[8]=8**
   3) **A[0]=1 A[1]=2 A[2]=3 A[3]=4 A[4]=5 A[5]=6 A[6]=7 A[7]=8**
20. Что нужно изменить в операторе **for(i=0; i<k; i++) cout<<A[i]<<endl;** чтобы массив **A[k]** был выведен на экран в виде строки?
   1) Вместо **i** поставить **K** 2) Убрать **<<endl** 3) Этого нельзя сделать