Тема: Массивы (табличные величины)

- Структурированные типы данных
- Основные понятия массивов
- Одномерные массивы
- Двумерные массивы

Структурированные (составные) типы данных

Структурированные (составные) типы данных - типы данных, состоящие из простых типов:

- **целых;**
- **вещественных;**
- **символьных;**
- **логических**

К таким типам данных относятся:

- Массивы и строки;
- Структуры и объединения;
- классы

Основные понятия массивов

Массив (табличная величина, таблица) – это упорядоченный набор фиксированного количества однотипных элементов, доступ к которым осуществляется с помощью индексов.

Массив – это последовательная группа ячеек памяти, имеющих одинаковое имя и одинаковый тип.

Имя массива – идентификатор.

Размер массива – количество элементов, т.е. число ячеек памяти.

Основные понятия массивов

Элементы массива называются индексными элементами (переменными с индексом).

Все элементы массива имеют порядковый номер – индекс. Индекс может быть - переменной, константой, ариф. выражением целого типа. Индексация элементов с нуля.

По количеству индексов, которые нужны для доступа к конкретному элементу массива, разделяют одномерные, двумерные, ***n***-мерные массивы.

Основные понятия массивов

Одномерный массив (линейная таблица, вектор) – это массив, каждый элемент которого определяется одним индексом.

Элементы одномерного массива в памяти компьютера располагаются последовательно.

Двумерный массив (матрица) – это массив, каждый элемент которого определяется двумя индексами.

Первый индекс указывает на номер строки, второй – на номер столбца в этой строке.

Одномерные массивы

Массивы занимают область в памяти. Программист указывает тип каждого элемента, количество элементов, требуемое каждым массивом, и компилятор может зарезервировать соответствующий объем памяти.

Объявление одномерного массива

```
<тип элементов> <имя массива>[<размерность>];
```

Например:

```
const int n = 10;
int A[n];
```

где n - его размер (именованная константа), число ячеек,
A – имя переменной, в которой хранится массив.

```
double b[100], x[27];
char s[30];  //хранение строки символов
```

Одномерные массивы

Инициализация одномерного массива

1 способ: при объявлении массива после знака = в фигурных скобках через запятую указать начальные значения

```
int n[5] = {1, 3, 5, 7, 11};
```

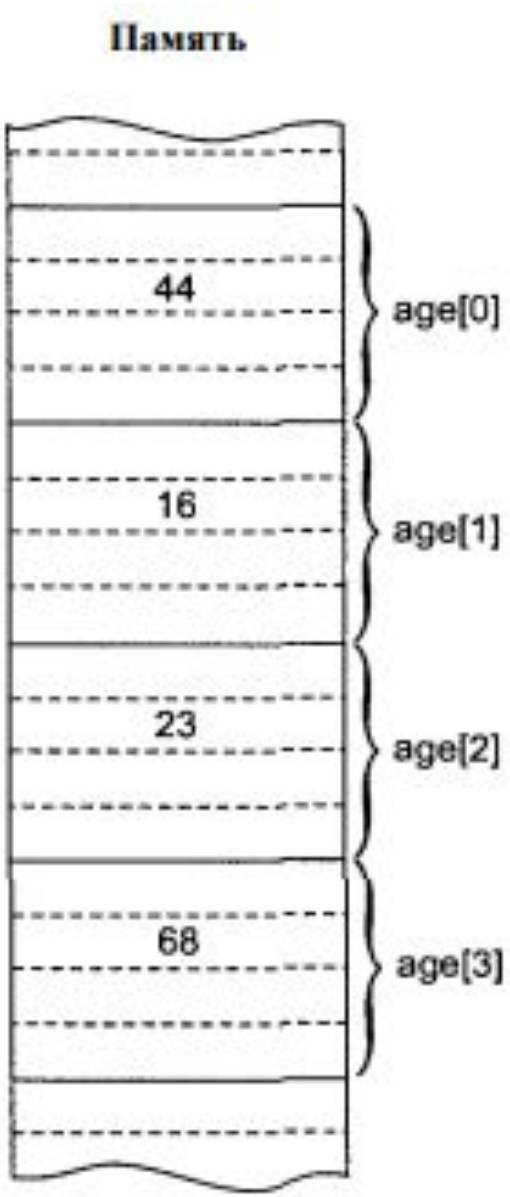
Если начальных значений больше, чем размерность массива - синтаксическая ошибка.

```
int n[5] = {32, 27, 64, 18, 95, 14};
```

Если начальных значений меньше, чем элементов в массиве, оставшиеся элементы автоматически получают нулевые начальные значения:

```
int n[10] = {1, 3, 5, 7, 11};
```

Одномерные массивы



Одномерные массивы

Обнуление элементов массива:

```
int n[10] = {0};
```

ЗАМЕЧАНИЕ:

1) явно присваивает нулевое начальное значение первому элементу и неявно – остальным элементам.

2) автоматически массивы не получают нулевые начальные значения неявно: нужно присвоить нулевое начальное значение по крайней мере первому элементу для того, чтобы автоматически были обнулены оставшиеся элементы.

Одномерные массивы

Если размер массива не указан в объявлении со списком инициализации, то количество элементов массива будет равно количеству элементов в списке начальных значений. Например:

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

Определение размера каждого массива с помощью именованной константы делает программу более масштабируемой.

```
const int m = 12;
int A[m];
```

Одномерные массивы

Индексация элементов массива

В операторах для обращения к i-ой ячейке (где 0 <= i < n) используется имя ячейки, например: `mas[i]`

где i - целое значение (или значение целой переменной, или целочисленного выражения), "индекс в массиве".

Эта операция [ ] называется "индексация массива".

Индексация - есть ВЫБОР одной из N ячеек при помощи указания целого номера.

mas - массив (n ячеек)

i - выражение (формула), выдающая целое значение в интервале 0..n-1, i – индекс элемента.

mas[i] - взят один из элементов массива.

Одномерные массивы

Доступ к элементам одномерного массива

<имя_массива>[индекс]

Например:

```
int mas[10], a, b;
b = 20;
mas[5] = 25;
a = mas[5] + b;
```

Вывод элементов одномерного массива

производится в цикле, обычно **for**

```
for (int i = 0; i < 10; i++)
  cout << mas[i] << '\n';//'\t'
```

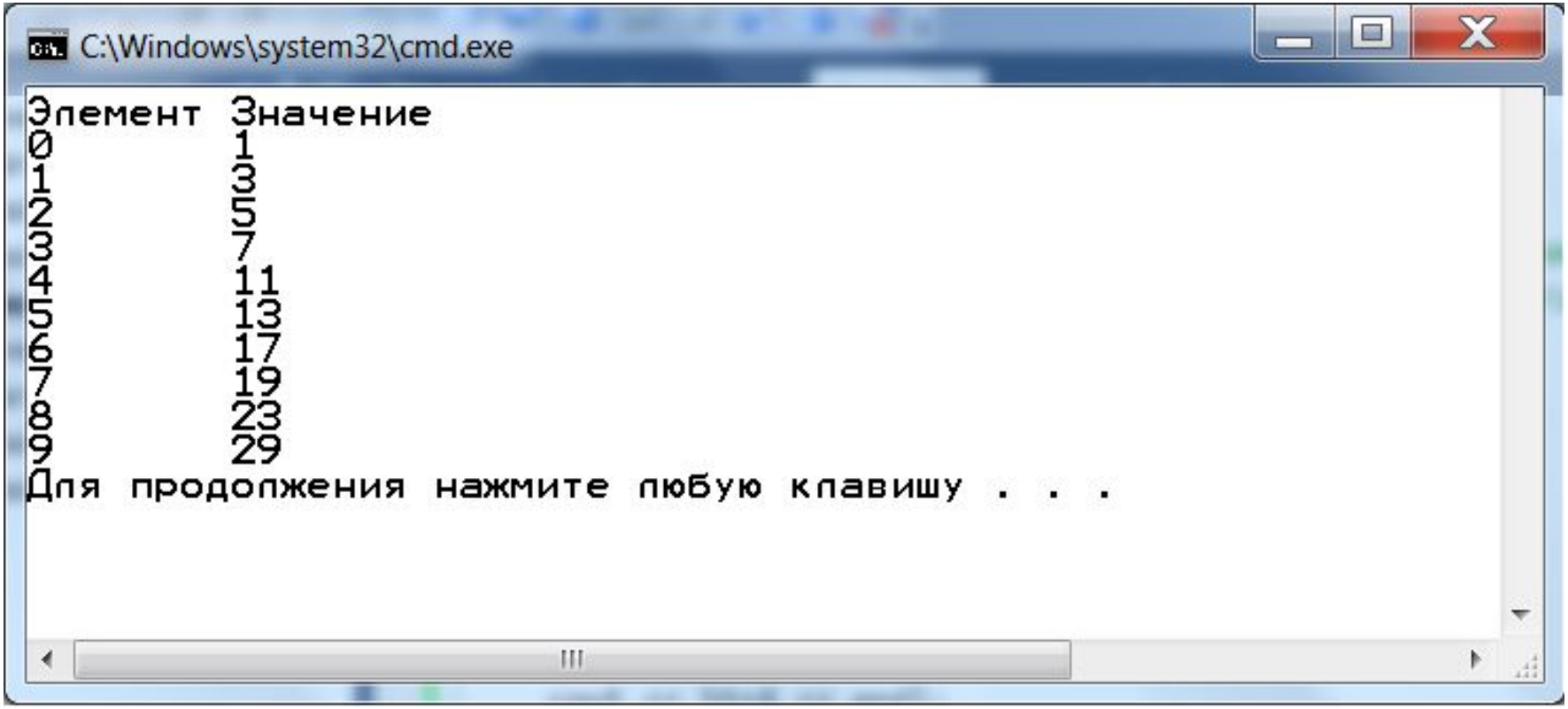
Одномерные массивы

Программа 1 присваивает десять начальных значений элементам массива целых чисел и печатает массив в табулированном формате

```
...
// объявление и инициализация массива
int mas[10] = {1, 3, 5, 7, 11, 13, 17,
19, 23, 29};

// печать массива
cout << "Элемент" << '\t' << "Значение"
<< '\n';
for (int i = 0; i < 10; i++)
cout << i << '\t' << mas[i] << '\n';
```

Одномерные массивы



Одномерные массивы

2 способ: используя индексацию, присвоить начальные значения уже в программе в цикле **for**. Причем значения элементов можно вводить с клавиатуры, вычислять по формуле или задавать с помощью генератора псевдослучайных чисел.

ЗАМЕЧАНИЕ: для получения целого числи из интервала [A..B]:

```
int x = A + rand()%(B+1-A);
```

- подключить библиотеку **stdlib.h**
- для инициализации генератора следует вызвать функцию **srand(time(0))**
- подключить библиотеку **time.h**

Одномерные массивы

Программа 2 заполняет массив из 20 элементов целыми случайными числами в диапазоне от -5 до 5 и выводит его на экран

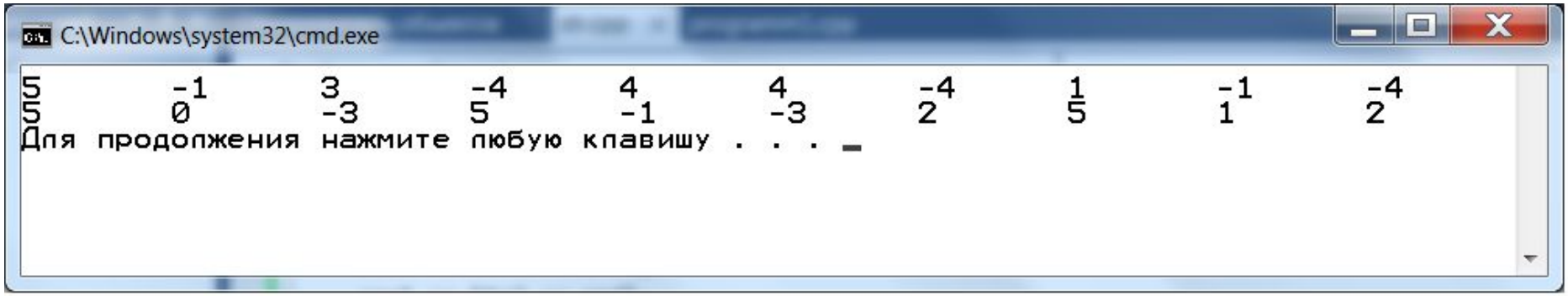
```
…
#include <stdlib.h>//библиотека для
          //использования функции
//генерирования случайных чисел (ГСЧ)

#include <time.h>//библиотека для
                 //использования функции
               //инициализации ГСЧ
…
```

Одномерные массивы

```
…
srand(time(NULL));   //инициализируем
     // генератор случайных чисел
int mas[20];  //описываем массив из 20
          //целых элементов
for (int i = 0;i < 20;i++)
{
//в цикле заполняем случайными числами
     //от -5 до 5
   mas[i]=rand()%11-5; //cin >> mas[i];
   cout << mas[i] << "\t";//сразу выводим
   //элементы массива через табуляцию
}
…
```

Одномерные массивы



Одномерные массивы

Инициализация строк

Строка всегда заканчивается 0.

1 способ (объявление и инициализация):

`char Str1[20] = "abcdefghij";` - строка будет проинициализирована значением, стоящим в кавычках, и, конечно, последний символ 0.

`char Str2[5] = "abcde";` - даст ошибку, потому, что всегда в этом инициализаторе последний символ 0, всего 6.

`char Str3[] = "abcde";`- определит размер 6

`char Str4[30] = "";` - пустая строка

Одномерные массивы

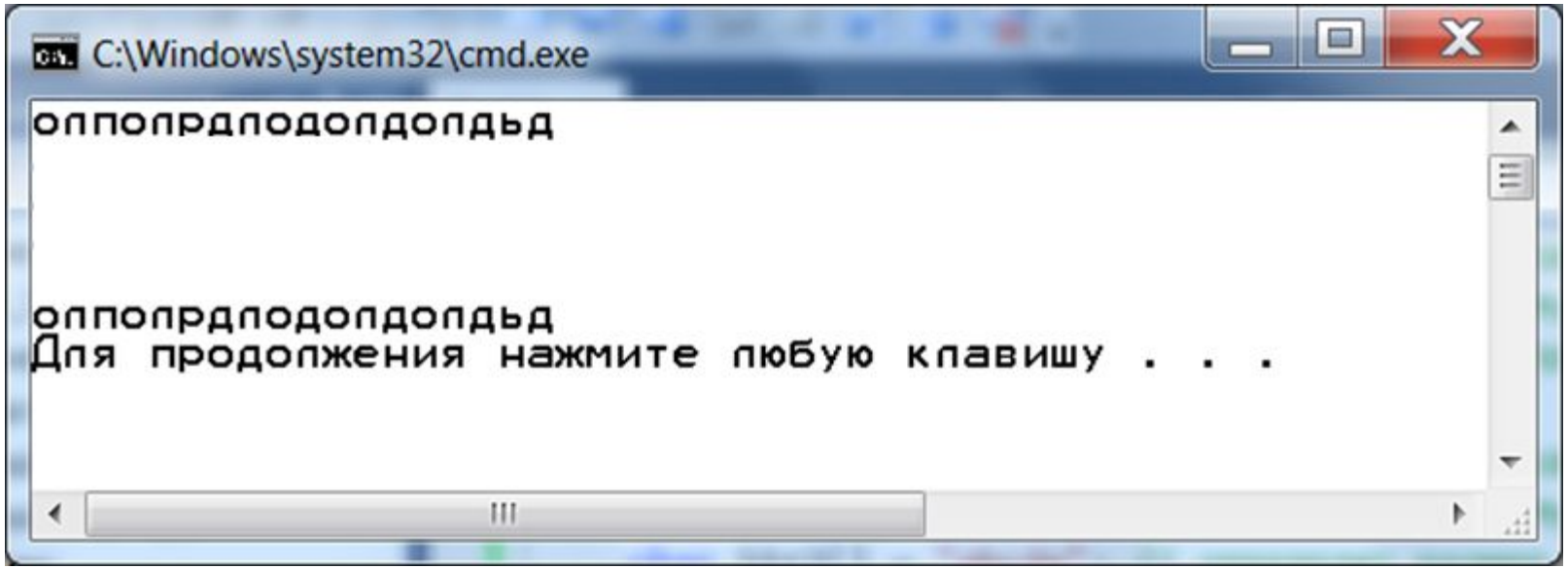
2 способ (объявление и ввод с клавиатуры)

```
char Str5[20];
cin >> Str5;
```

Вывод строки:

```
cout << имя массива типа char;
```

Например: `cout << Str1 << endl;`



Двумерные массивы

Объявление двумерного массива

```
<тип элементов> <имя массива>[<количество строк>][<количество столбцов>];
```

Например:

```
int a[2][3];
double b[50][50], c[10][10];
char Trio[100][30];// одномерный массив
        // строк, состоящий  из 100
// строк, каждая длинной 30.
```

ЗАМЕЧАНИЕ: размер строк и столбцов можно указывать через именованные константы

Двумерные массивы

Инициализация двумерного массива

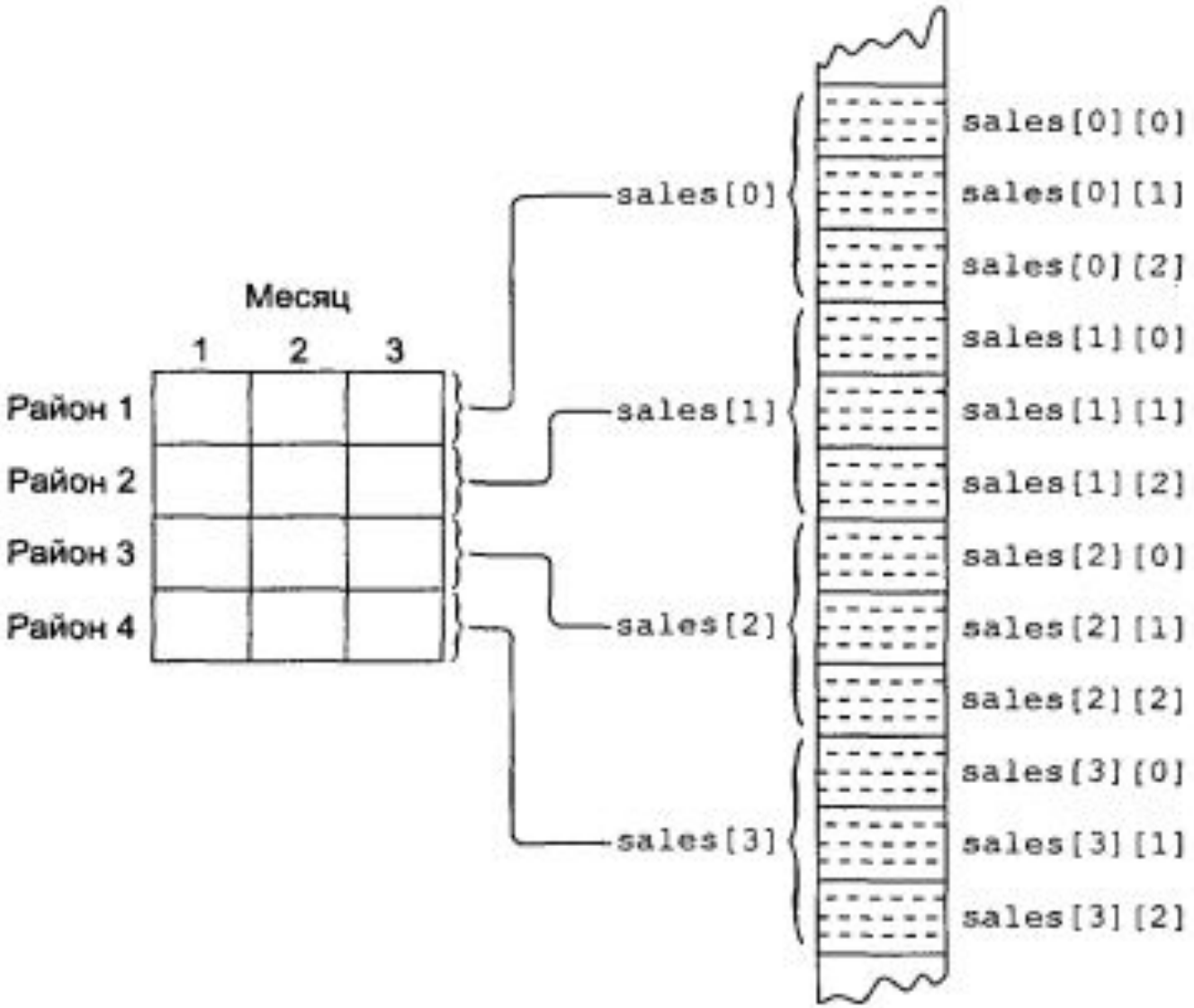
1 способ: при объявлении массива после знака = в фигурных скобках через запятую сгруппировать в фигурных скобках строки массива

```
int B[2][2]={{1,2},{3,4}};
```

Если начальных значений больше, чем размерность массива - синтаксическая ошибка.

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

Двумерные массивы



Двумерные массивы

Если начальных значений меньше, чем элементов в массиве, оставшиеся элементы автоматически получают нулевые начальные значения:

```
int B[2][2]={{1},{3,4}};
```

Обнуление элементов массива:

```
int n[10][5] = {0};
```

Двумерные массивы

Доступ к элементам двумерного массива

<имя_масс>[номер стр.][номер стб.]

Например:

```
int mas[10][5], a, b;
b = 20;
mas[5][1] = 25;
a = mas[5][1] + b;
```

ЗАМЕЧАНИЕ: индексация строк и столбцов начинается с нуля

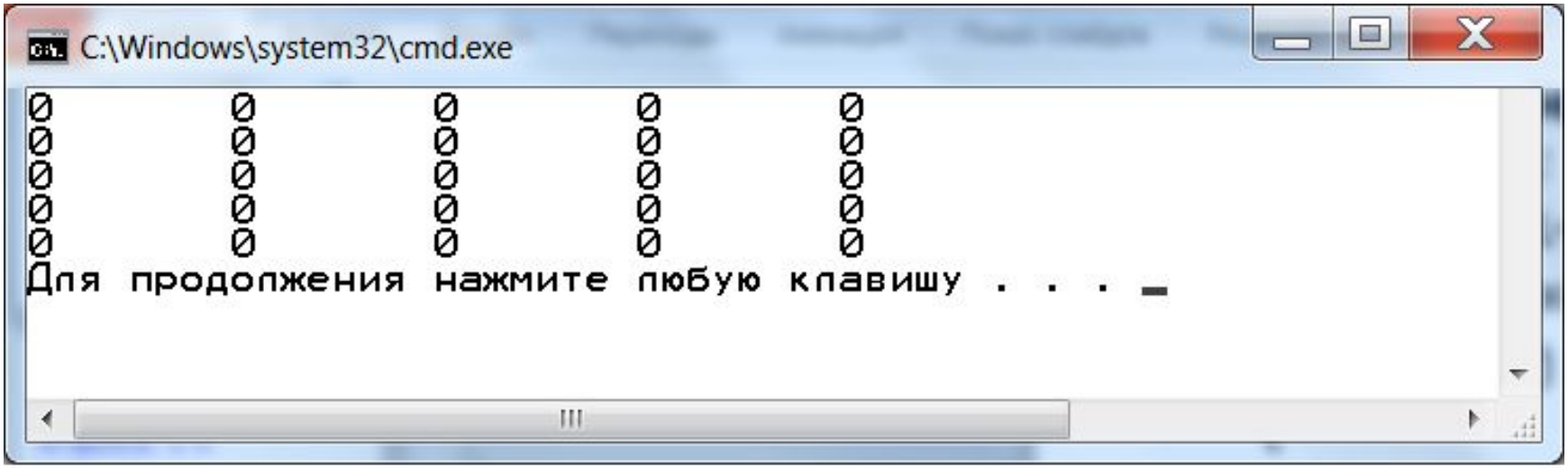
Двумерные массивы

Вывод элементов двумерного массива

производится во вложенных циклах **for:** внешний цикл - строки, внутренний - столбцы. Индексация – с нуля.

```
int mas[5][5];
for (int i=0;i<5;i++)
{ //вывод на экран элементов матрицы построчно
for(int j =0;j<5;j++)
   {
      mas[i][j]=0;
//во внутреннем цикле заполняем матрицу нулями
   cout<<mas[i][j]<<"\t"; //выводим на
                          //экран
   }
   cout<<"\n"; //закончив строку - ENTER
}
```

Двумерные массивы



Двумерные массивы

2 способ: используя индексацию, присвоить начальные значения уже в программе в циклах **for**.

Причем значения элементов можно вводить с клавиатуры, вычислять по формуле или задавать с помощью генератора псевдослучайных целых чисел.

Двумерные массивы

Инициализация одномерного массива строк

1 способ (объявление и инициализация):
при объявлении массива строк после знака = в фигурных скобках через запятую указать в двойных кавычках строки массива

```
char Trio[4][30] = {"abcdfe",
"ghijklm", "nopqrstuvw", "xyz"};
```

Двумерные массивы

2 способ (объявление и ввод с клавиатуры)

```
char Trio[4][30];
for (int i = 0;i < 4; i++)
   cin >> Trio[i];
```

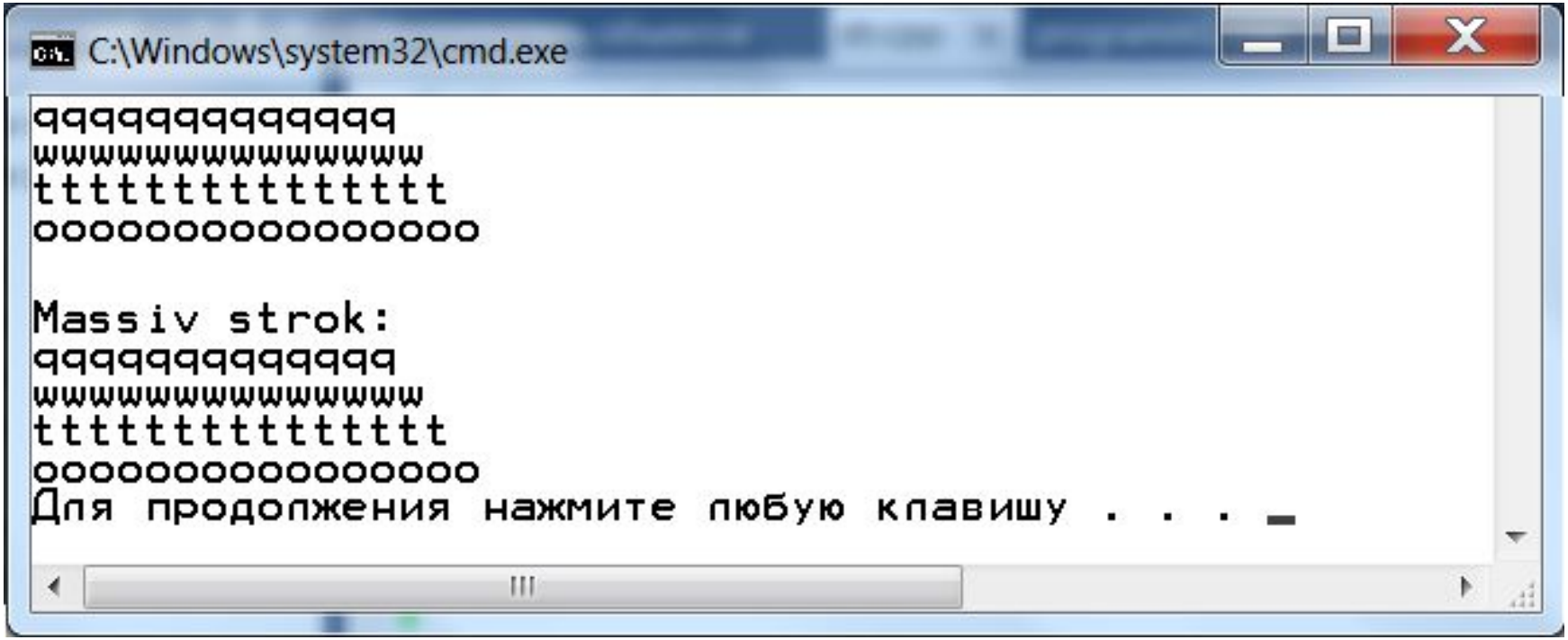
Вывод строки:

Вывод в цикле по строкам в операторе cout каждой строки.

Например:

```
cout << "Massiv strok: \n";
for (int i = 0;i < 4; i++)
   cout << Trio[i] << endl;
```

Двумерные массивы



Строки

Представление строк на C++

Строки в C++ представляются массивами символов.
char Save[10] – можно считать, что это строка.
c
Чтобы обнулить строку достаточно в нулевой элемент массива записать 0.
Благодаря этому следующая программа распечатает всю память, находящуюся за пределами строки, до тех пор, пока не встретит 0.

```
int D[4] = {0x28282828, 0x28282828, 0x28282828, 0x28282828};
char S[3] = {48, 49, 50};
cout<<S;
```

Для того чтобы куда либо передать строку достаточно указать её начало.
Массивы символов имеют особое значение, по умолчанию подразумевается, что массив символов это строка.

Строки

Обычно в **C++** используются два вида строк**:**

1) строка как массив символов типа **char** **(**можно назвать строковым типом или этот тип можно также назвать с**h**аг***-**строками**,** так как он может быть представлен в виде указателя на **char (*** означает указатель**).**

Строки

2) строка как объект класса **string.**

Строки, созданные с помощью класса **string,** во многих ситуациях вытеснили строковый тип.

Но строковый тип все еще используется,т.к.:

- ✔ Во-первых, он используется во многих библиотечных функциях языка **C.**
- ✔ Во-вторых, он год за годом продолжает появляться в кодах.
- ✔ В-третьих, для изучающих **C++** строковый тип наиболее примитивен и поэтому легко понимается на начальном уровне.

Строки

Рассмотрим строку, как массив элементов типа **char** – строковый тип.

char Save[10];

– можно считать, что это строка.

Каждый символ занимает **1** байт памяти. Строки должны завершаться байтом, содержащим **0**.
Это часто представляют символьной константой **\0**, код которой в **ASCII** равен **0**.
Завершающий ноль называется **нулевым символом**.

Строки

```
char dog[5] = { 'b', 'e', 'a', 'u', 'x'};
          // не строка!
char cat[5] = {'f', 'a', 't', 's', '\0'};
          // строка!
```

Строки в кавычках всегда неявно включают конечный нулевой символ, поэтому писать его не надо

```
char boss[8] = "Bozo";
```

Строки

Как и другие типы данных, строки могут быть переменными и константами.

Строковые переменные

Объявление строковой переменной:

```
const int n = 10;
char str[n];
```

Инициализация:

```
cin >> str;
```

Вывод на экран:

```
cout << str;
```

Строки

Пример. Программа просит пользователя ввести строку и помещает эту строку в строковую переменную, а затем выводит эту строку на экран.

```
// простая переменная строка
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////
int main()
{
   const int MAX = 80;    // максимальный размер строки
   char str[MAX];                          // сама строка
   cout << "Введите строку: ";             // ввод строки
   cin >> str;
   cout << "Вы ввели: " << str << endl;// показ результата
   return 0;
}
```

Строки

Когда операция << выводит строку, то она выводит символы до тех пор, пока не встретит нулевой символ.

Рис. Строка, хранящаяся в строковой переменной

Строки

Как избежать переполнения буфера?

Что случится, если пользователь введет строку, которая окажется длиннее, чем массив, используемый для ее хранения?
В C++ нет встроенного механизма, защищающего программу от помещения элементов за пределы массива.
Поэтому слишком увлеченный пользователь может привести систему к краху.

Однако существует возможность ограничить при использовании операции **>>** количество символов, помещаемых в массив.

Строки

```
cin >> setw(n) >> str5; // ввод не
        // более чем n символов
```

В операторе ввода используется метод **setw**, определяющий максимальное количество символов, которое сможет принять буфер. Пользователь может напечатать больше символов, но операция **>>** не вставит их в массив.

В действительности операция вставит в массив на один символ меньше, чем определено, так как в буфере есть место для завершающего нулевого символа, т. е. здесь максимальное число возможных символов 19.

Строки

Строковые константы

Позволяют проинициализировать строку постоянным значением при ее определении.

Объявление строковой константы:

```
1) char str[] = “Hello!”;
```

ЗАМЕЧАНИЕ:

```
char str[] = {'H','e','l','l','o','!'\0'};

2) char Str2[20] = "abcdefghij";
3) char Str3[30] = "";
```

Вывод на экран:

```
cout << str;
```

Строки

Чтение внутренних пробелов

В случае, если строковая переменная содержит более одного слова (при инициализации в операторе **cin**), то при выводе строки в операторе **cout** будет выведено только первое слово.

Для считывания строк, содержащих пробелы, мы используется метод
cin.get()

Этот синтаксис означает использовать метод **get()** класса **stream** для его объекта **cin**.

```
// ввод строки с пробелами
#include <iostream>
using namespace std;
////////////////////////////////////////////////////
int main()
{
   const int MAX = 80; // максимальная длина строки
   char str[MAX];      // сама строка

   cout << "\nВведите строку: ";

   cin.get(str, MAX);
   cout << "Вы ввели: " << str << endl;
   return 0;
}
```

Первый аргумент метода **cin.get()** — это адрес массива, куда будет помещена введенная строка. Второй аргумент определяет максимальный размер массива, автоматически предупреждая, таким образом, его переполнение.

Строки

Считывание нескольких строк (без использования одномерного массива строк)

Метод **cin.get()** может иметь третий аргумент, который определяет символ, на котором метод завершает считывание строки.
Установленным по умолчанию значением этого аргумента является символ новой строки ('**\n**'), но если вызвать метод с другим аргументом, то это значение заменится на введенный вами символ.

```
// ввод нескольких строк
#include <iostream>
using namespace std;
const int MAX = 2000; // максимальная длина строки
char str[MAX];        // сама строка
//////////////////////////////////////////////////
int main()
{
   cout << "\nВведите строку:\n";
   cin.get(str, MAX, '$');
   cout << "Вы ввели:\n" << str << endl;
   return 0;
}
```

Метод будет принимать символы до тех пор, пока не введете завершающий символ (или до тех пор, пока введенные данные не превысят размер массива). Нужно будет нажать клавишу Enter после того, как будет напечатан символ '$'

```
Введите строку:
Широка страна моя родная
Много в ней лесов, полей и рек
Я другой такой страны не знаю.
Где так вольно дышит человек.
$
Вы ввели:
Широка страна моя родная
Много в ней лесов, полей и рек
Я другой такой страны не знаю,
Где так вольно дышит человек.
```

ЗАМЕЧАНИЕ: Мы заканчивали каждую строку нажатием клавиши Enter, но программа продолжала принимать от нас ввод до тех пор, пока мы не ввели символ '$'.

Строки

Операции со строками:

-Копирование

-Добавление строки к строке

-Сравнение

-Определение длины строки

-Поиск символа в строке в строке

-Изменение регистра букв

-Изменение кодировки букв и т.д.

Строки

- Производя какую-нибудь операцию со строками, приходится работать с каждым её символом.

Для копирования из одной строки в другую следует копировать каждый её символ, тоже самое при сравнении строк.

Обычно используется цикл.

- **Операции со строками можно производить с использованием стандартных функций для работы со строками: заголовочный (библиотечный) файл<cstring> (в старых версиях <string.h>)**

Строки

Определение длины строки **strlen**

Формат

```
strlen(строка, длину которой считают);
```

Пример:

```
char Src[20] = "string";
strlen(Src);
```

Результат

6 – длина строки

ЗАМЕЧАНИЕ: Однако длина строки, возвращаемая функцией strlen(), не включает в себя нулевой символ.

Строки

Копирование строк

1 способ (копирование с использованием цикла).

```
// исходная строка
char str1[] = "Маленькой елочке холодно зимой,";
const int MAX = 80;                          // максимальная длина строки
char str2[MAX];                              // сама строка
for(int j = 0; j < strlen(str1); j++) // копируем strlen(str1)
   str2[j] = str1[j];                           // символов из str1 в str2
str2[j] = '\0';                              // завершаем строку нулем
cout << str2 << endl;                       // и выводим на экран
```

и строковую переменную str2. Затем в цикле for происходит копирование строковой константы в строковую переменную. За каждую итерацию цикла копируется один символ в строке

Строки

```
int count = 0;
while (true) // запускаем бесконечный
          // цикл
{
  str2[count] = str1[count]; //копируем
                //посимвольно
  if (str1[count] == '\0') // если
       //нашли \0 у первой                    //
строки
  {
  break; // прерываем цикл
  }
  count++;
}
```

Строки

2 способ (копирование с использованием функции **`strcpy()`**).

```
char str1[] = "Уронили мишку на пол, оторвали мишке лапу!";
const int MAX = 80:    // максимальная длина строки
char str2[MAX];        // сама строка
strcpy(str2, str1);    // копируем строку
cout << str2 << endl; // и показываем результат
```

Заметим, что первым аргументом этой функции является строка, куда будут копироваться данные:

`strcpy(destination, source);`

Порядок записи справа налево напоминает формат обыкновенного присваивания: переменная справа копируется в переменную слева.

Строки

Копирование строк. **strncpy не до конца строки а до указанного числа символов**

Формат

strncpy(*строка в которую копируют, строка которую копируют, длинна строки*);

Пример

```
char Src[20] = "copy string";
char Dst[20];
strncpy(Dst, Src, 7);
```

Результат

```
Src -> "copy string";
Dst -> "copy st";
```

Строки

Добавление строки к строке

1 способ (копирование с использованием цикла).

```
char Src[20] = "adding string";
char Dst[20] = "base string";
// поиск конца строки
for(int n = 0; Dst[n]; n++);
// теперь копирование
for(int m = 0; Src[m]; m++)
  Dst[n + m] = Src[m];
Dst[n + m] = Src[m];
```

Строки

2 способ (с использованием функций strcat и strncat)

Формат

strcat(строка в которую добавляют, строка которую добавляют);

Пример

```
char Src[20] = "adding string";
char Dst[20] = “base string”;
strcat(Dst, Src);
```

Результат

```
Src -> “adding string”;
Dst -> “base stringadding string”;
```

Строки

Сравнение строк strcmp и strncmp

Формат

strcmp(строка1 которую сравнивают, строка2 которую сравнивают);

Пример

```
char Str1[20] = "string1";
char Str2[20] = “string2”;
strcmp(Str1, Str2);
```

Результат

-1 - первая строка меньше второй

+1 - первая строка больше второй

0 – строки равны

Строки

Поиск подстроки в строке strstr

Формат

strstr(строка в которой ищут, строка которую ищут);

Пример

```
char Str[20] = "string lord";
char FStr[20] = “ring”;
strstr(Str, FStr);
```

Результат

0 – строка не найдена

не 0 – найдена – начало найденной строки

Строки

Поиск символа в строке в строке strchr

Формат

strchr(строка в которой ищут, символ который ищут);

Пример

```
char Str[20] = "string lord";
strchr(Str, 't');
```

Результат

0 – символ не найден

не 0 – найден – начало найденного символа

Строки

Все буквы в строке сделать большими strupr

Формат

strupr(строка в которой изменяют буквы);

Пример

```
char Src[20] = "string";
strupr(Src);
```

Результат

Src -> "STRING";

Строки

Все буквы в строке сделать маленькими strlwr

Формат

strlwr(строка в которой изменяют буквы);

Пример

```
char Src[20] = "STrinG";
strlwr(Src);
```

Результат

Src -> "string";

Строки

Все буквы в строке сделать маленькими strlwr

Формат

```
strlwr(строка в которой изменяют буквы);
```

Пример

```
char Src[20] = "STrinG";
strlwr(Src);
```

Результат

Src -> "string";

Строки

Перевод русского текста из DOS кодировки в WINDOWS и OemToCharA и OemToCharBuffA (windows.h)

Формат

OemToCharA (строка из которой переводят, строка в которую переводят);

Пример

```
char Src[20] = "русский текст";
char Dst[20];
OemToCharA(Src, Dst);
```

Результат

Буквы русского алфавита переведены в WINDOWS формат.

Строки

Перевод русского текста из WINDOWS кодировки в DOS и CharToOemA и CharToOemBuffA

Формат

CharToOemA (строка из которой переводят, строка в которую переводят);

Пример

```
char Src[20] = "русский текст";
char Dst[20];
CharToOemA(Src, Dst);
```

Результат

Буквы русского алфавита переведены в DOS формат.

Строки

Инициализация одномерного массива строк

1 способ (объявление и инициализация):
при объявлении массива строк после знака = в фигурных скобках через запятую указать в двойных кавычках строки массива

```
char Trio[4][30] = {"abcdfe",
"ghijklm", "nopqrstuvw", "xyz"};
```

Строки

2 способ (объявление и ввод с клавиатуры)

```
char Trio[4][30];
for (int i = 0;i < 4; i++)
   cin >> Trio[i];
```

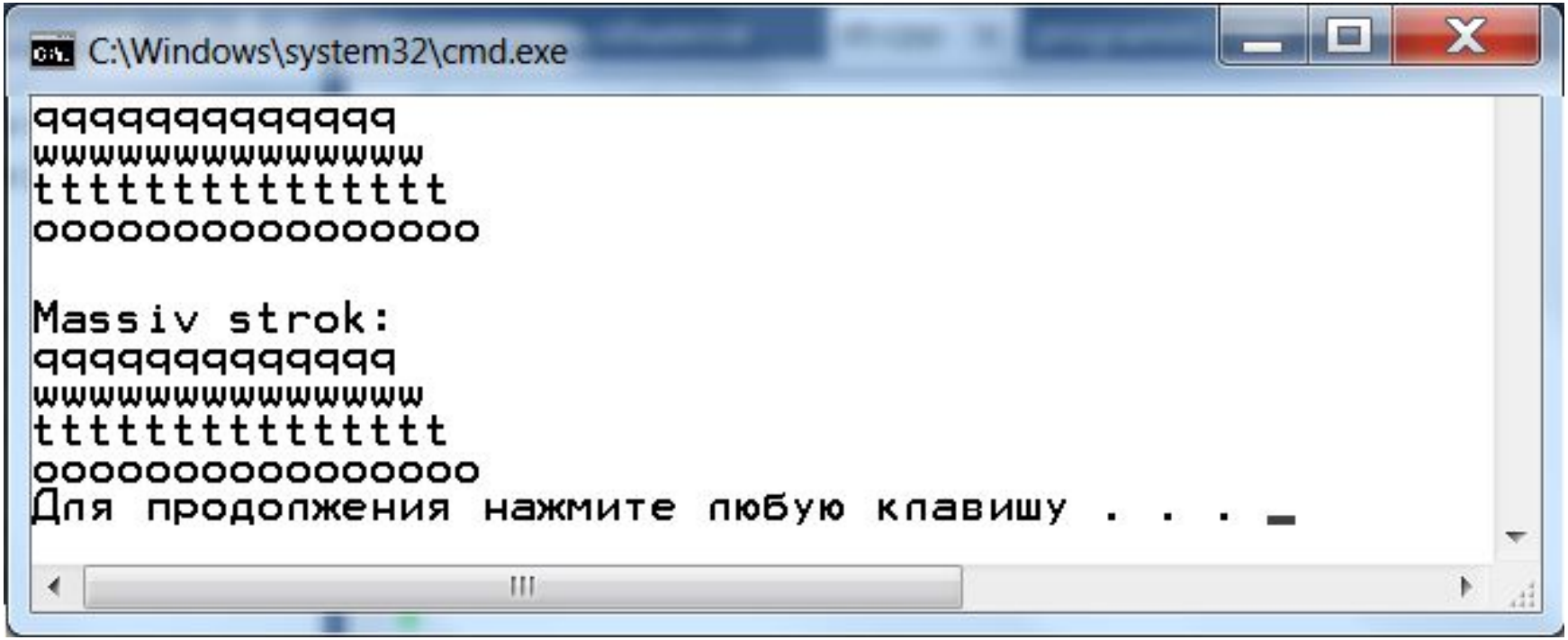
Вывод строки:

Вывод в цикле по строкам в операторе cout каждой строки.

Например:

```
cout << "Massiv strok: \n";
for (int i = 0;i < 4; i++)
   cout << Trio[i] << endl;
```

Строки


C:\Windows\system32\cmd.exe
qqqqqqqqqqqqq
wwwwwwwwwwwwww
ttttttttttttt
ooooooooooooooo
Massiv strok:
qqqqqqqqqqqqq
wwwwwwwwwwwwww
ttttttttttttt
ooooooooooooooo
Для продолжения нажмите любую клавишу . . .

Обработка элементов массива

- Поиск элементов, удовлетворяющих условию (например, макс или мин)
- Вычисление суммы, произведения и т.д. элементов
- Упорядочивание элементов массива

Сортировка массивов

Сортировка – это упорядочение данных по какому-либо признаку. Чаще всего сортировка связана с упорядочением элементов массива по возрастанию и убыванию.

Методы сортировки

- Сортировка пузырьком
- Сортировка вставками
- Сортировка выбором
- Сортировка слиянием
- Быстрая сортировка

Сортировка массивов

1. Метод «Пузырька» («Пузырьковая» сортировка) - подробно пузырьку, больший элемент массива поднимается "вверх".

Алгоритм:

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов.

Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован.

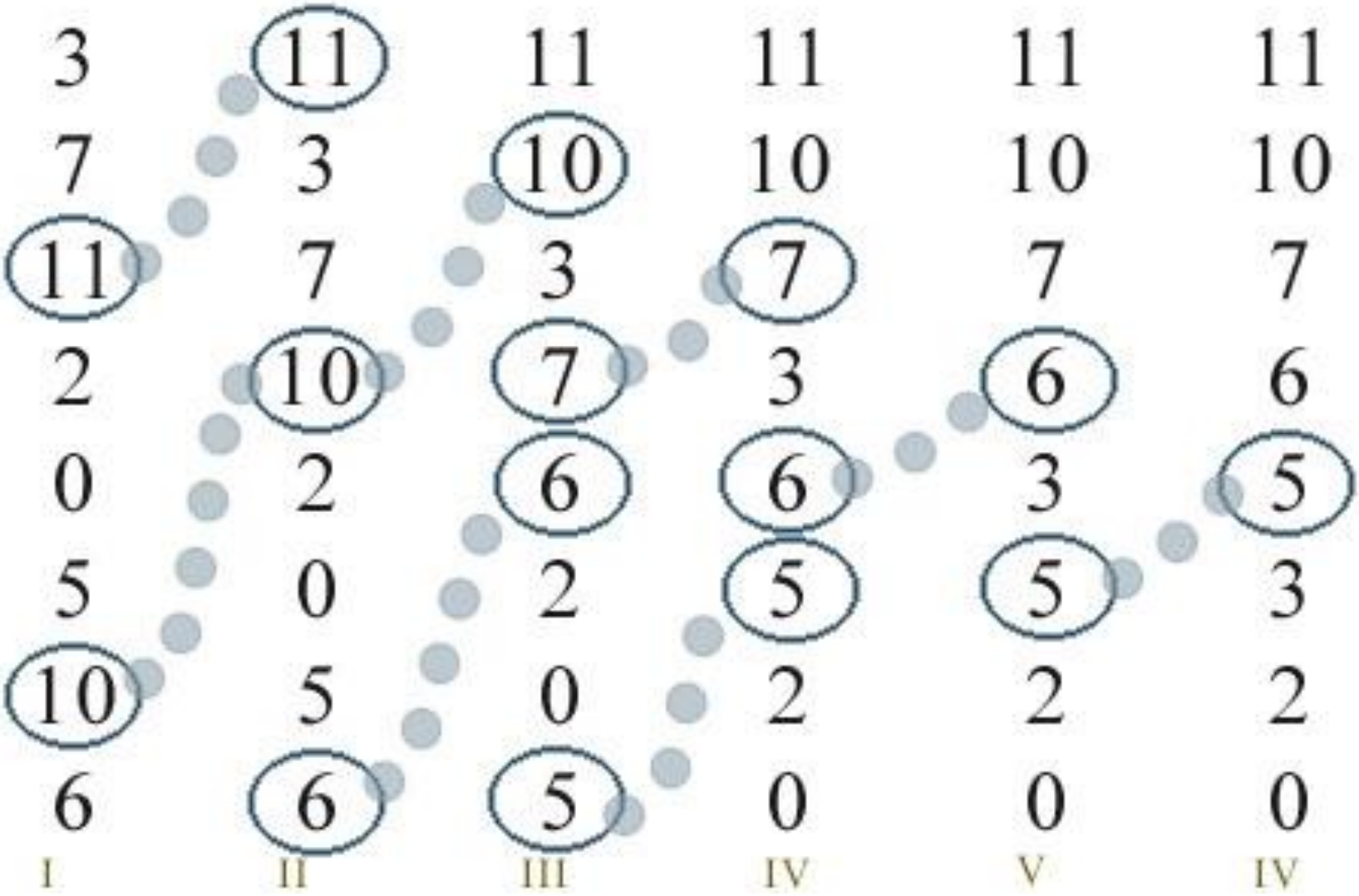
Сортировка массивов

При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива

Достоинство пузырьковой сортировки - простоте ее программирования.

Недостатки - выполняется она медленно. Это становится очевидным при сортировке больших массивов.

Сортировка массивов

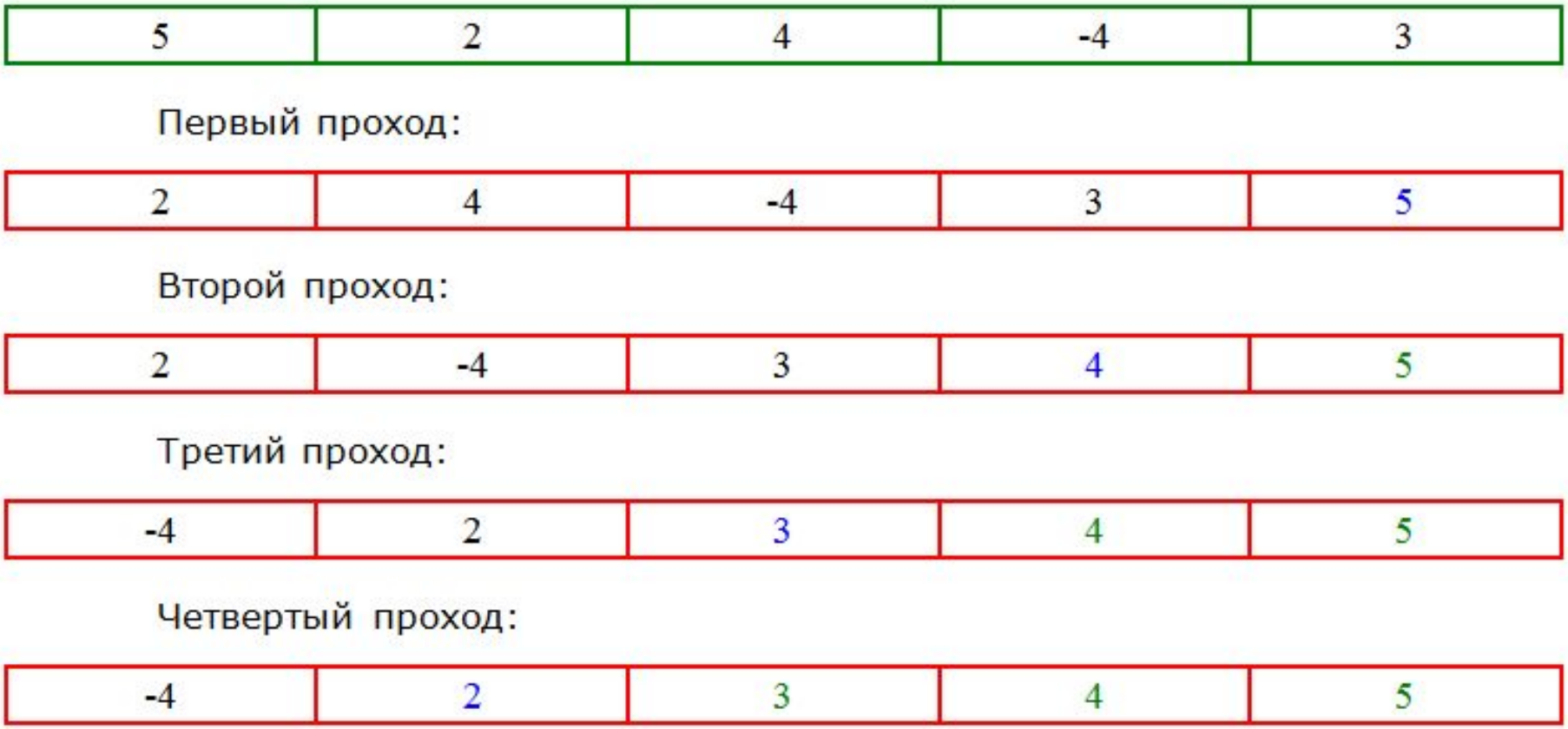


Сортировка массивов

Программа сортирует значения массива следующим образом: сначала она сравнивает элементы массива mas[0] и mas[1], меняя местами их значения, если они не упорядочены, затем проделывает то же самое с элементами mas[1] и mas[2], mas[2] и mas[3] и т.д.
При выполнении этой последовательности операций элементы с большим значением будут продвигаться вправо, и на самом деле элемент с наибольшим значением займет положение mas[7]. При многократном выполнении этого процесса соответствующие элементы попадут в позиции mas[6], mas[5] и т.д., так что в конце концов все элементы будут упорядочены.

Сортировка массивов

Пример



Замечание: сортировка в любом случае выполнится за n-1 проход.

Сортировка массивов

"Пузырьковая" сортировка вектора по убыванию

```
const int n = 5; //размерность массива
int mas[n] = {5, 2, 4, -4, 3};
int buffer;  //переменная - буфер
     // обмена
int i,j; //счетчики циклов

//вывод исходного массива в столбец
for (i = 0; i < n; i++)
     cout<<"element " << i <<
          ":" << mas[i] << "\n";
   cout <<"\n";
```

```
/*внешний цикл до n - 1  (чтобы избежать
заведомо лишнего прохода Назначение:
регулирует количество проходов по массиву */
for(i = 0; i < n - 1; i++)
   /*внутр. цикл до n - 1 – i (т.к.
сравниваются элемент j и j + 1, а так же –i,
чтобы не проверять уже отсортированные
элементы на i-х проходах)
Назначение: сорт. элементы в i-м проходе */
      for(j = 0; j < n – 1 - i; j++)
//замена элементов местами с помощью б.о
if (mas[j] < mas[j + 1])
     { buffer = mas[j];
mas[j] = mas[j + 1];
        mas[j + 1] = buffer;
     }
```

```
//вывод в столбец отсортированного масс
for(i = 0; i < n; i++)
   cout<<"elem "<<i<<": " << mas[i] <<
      "\n";
cout<<"\n";
```

Альтернативный вариант

```
for (i = n - 1; i >= 0; i--)
{
   for (j=0; j < i; j++)
   {
      if (mas[j] < mas[j+1])
       { …  }
       }
}
```

Сортировка массивов

Сортировка строк пузырьками

```
const int n = 6;
char mass[n][10] = {"sergey",
"nikolay", "aleksandr", "alexey" ,
"semen", "vlad"};
int i,j;
for(i = 0; i < n; i++)
    for(j = 0; j < n - 1 - i; j++)
      if (strcmp(mass[j],mass[j + 1]) > 0)
      {
        char t[10];
        strcpy(t, mass[j]);
        strcpy(mass[j], mass[j + 1]);
        strcpy(mass[j + 1], t);
      }
```

Сортировка массивов

2. Сортировка выбором - упорядочиваем постепенно массив, заполняя первую позицию неупорядоченной части минимальным элементом из неупорядоченной части.

Алгоритм

1. Находим номер минимального значения в текущем списке
2. Производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции)
3. Теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы.

Сортировка массивов

Пример

Сортировка массивов

-4	2	3	5	4

Четвертый проход:

-4	2	3	5	4
			i	min

Замена:

-4	2	3	4	5

Массив отсортирован

Сортировка массивов

Пусть в массиве mas находится N элементов, тогда для i от 0 до N - 1 необходимо выполнить следующие шаги:

Считаем, что i-ый элемент цепочки является минимальным среди элементов, идущих после него.

Для элементов от mas[i + 1] до mas[N - 1] выполнить сравнение по ключам и найти наименьший элемент. Назовем его mas[min].

Поменять местами mas[min] и mas[i]. mas[min] теперь находится в своей отсортированной позиции.

После окончания цикла массив отсортирован.

Сортировка массивов

Сортировка вектора выбором

```
// Счетчики циклов
int i, j;
/* Временная переменная для обмена
значений элементов  */
int buffer;
/* Переменная для хранения индекса
минимального элемента в
последовательности*/
int min;
const int n = 6; //размерность массива
int mas[n];
```

Сортировка массивов

```
for(i = 0; i < n - 1; i++)
{    // На каждой итерации определяем
элемент с индексом i
// как минимальный
   min = i;
   for(j = i + 1; j < n; j++)
   { // Поиск минимального элемента
   if(mas[j] < mas[min])                min =
j;
   }
```

Сортировка массивов

```
// Если переменная min не изменилась,
то i-ый элемент
// находится на своем месте
if(min == i) continue;    // Иначе
меняем i-ый и найденный (меньший)
элемент местами
     buffer = mas[i];
     mas [i] = mas[min];
mas[min] = buffer;
   }
}
```

Сортировка массивов

Сортировка строк вставками:

```
const int max_elem = 6;
char mass[max_elem][10] = {"sergey",
"nikolay", "aleksandr", "alexey" ,
"semen", "vlad"};
int elems = max_elem;
int n;
```

Сортировка массивов

```
for(int cur = 0; cur < elems; cur++)
  {
    int nmin = cur;
    for(int n = cur; n < elems; n++)
      if (strcmp(mass[nmin],mass[n])>0)
        nmin = n;
    char t[10];
    strcpy(t, mass[nmin]);
    strcpy(mass[nmin], mass[cur]);
    strcpy(mass[cur], t);
  }
```

Сортировка массивов

3. Метод вставок - выбираем и вставляем элемент в нужную позицию.

Алгоритм

На каждом шаге алгоритма мы выбираем один из элементов входных данных и вставляем его на нужную позицию в уже отсортированном списке, до тех пор, пока набор входных данных не будет исчерпан. Метод выбора очередного элемента из исходного массива произволен; может использоваться практически любой алгоритм выбора. Обычно (и с целью получения устойчивого алгоритма сортировки), элементы вставляются по порядку их появления во входном массиве.

Сортировка массивов

Вставить пример!

Альтернативный вариант

```
int i, j, value;
for(i = 1; i < n; i++)
{
    value = a[i];
    for (j = i - 1; j >= 0 && a[j] > value; j--)
    {
        a[j + 1] = a[j];
    }
    a[j + 1] = value;
    }
```