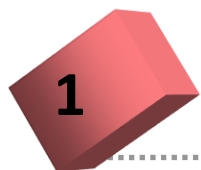




ФУНКЦИИ

**Дальневосточный государственный университет путей сообщения,
кафедра «Вычислительная техника и компьютерная графика»,
к.т.н., доцент Белозеров Олег Иванович.**

Вопросы лекции:



Указатели и ссылки




Фактические и формальные параметры функции



Основные методы сортировки

Что получим при выполнении программы?



```
#include <iostream>

int main()
{
    int g = '8' - '4';
    std::cout << g;
}
```

1. Указатели и ссылки

Когда компилятор обрабатывает оператор определения переменной, например, *int a = 50;*, то он выделяет память в соответствии с типом *int* и записывает в нее значение 50.

Все обращения в программе к переменной по ее имени заменяются компилятором на адрес области памяти, в которой хранится значение переменной, **такие переменные называются указателями.**

В С++ различают три вида указателей:

указатели на объект, на функцию и на void.

Указатель на объект содержит адрес области памяти, в которой хранятся данные определенного типа (простого или составного).

Объявление указателя на объект имеет следующий вид:

<базовый тип> [<модификатор>] * <имя указателя>

базовый тип — имя типа переменной, адрес которой будет содержать переменная указатель;

модификатор необязателен., может иметь значение: *near*, *far* или *huge*.

Оператор &. Это унарный оператор, возвращающий адрес операнда в памяти. Унарному оператору требуется только один операнд. Например:

```
m = &count;
```

помещает в `m` адрес переменной `count`. Это адрес внутреннего местоположения переменной в компьютере.

С самим значением переменной ничего не делается. Оператор `&` можно запомнить как «взятие адреса». Поэтому вышеупомянутый оператор присваивания можно прочитать как «`m` получает адрес `count`».

Оператор *, дополняющий &. Это унарный оператор, возвращающий значение переменной по указанному адресу. Например: если `m` содержит адрес переменной `count`, то

`q = *m;`

помещает значение `count` в `q`.

Операция `*` может быть запомнена как «по адресу». В данном случае оператор можно прочесть как «`q` получает значение по адресу `m`».

К несчастью, значки для умножения и для взятия «по адресу» - одинаковы, впрочем как и значки битового И и «взятие адреса». Эти операторы не имеют связи друг с другом. Как `&` так и `*` имеют более высокий приоритет по сравнению с остальными арифметическими операциями, за исключением унарного минуса, имеющего такой же приоритет.

Переменные, содержащие адреса или указатели, должны объявляться путем помещения * перед именем переменной для указания компилятору того, что переменная содержит указатель. Например, для объявления указателя ch на символ, следует написать

```
char *ch;
```

Здесь ch - это не символ, а указатель на символ, в чем и заключается принципиальное различие. Тип данных, на который указывает указатель, как в нашем случае char, называется базовым типом указателя. Сам указатель - это переменная, используемая для хранения адреса объекта базового типа. Следовательно, указатель на символ (или любой другой указатель) имеет фиксированный размер, определяемый архитектурой компьютера, для хранения адреса.

Можно смешивать объявление указателей и обычных переменных в одной строке.

Например:

```
int x, *y, count;
```

объявляет `x` и `count` как переменные целочисленного типа, а `y` - как указатель на целочисленный тип.

Итак, давайте перечислим все по порядку. Указатель может быть переменной или константой, указывать на переменную или константу, а также быть указателем на указатель.

Например:

```
int i;           //целочисленная переменная
const int j=10;  //целочисленная константа
int *a;          //указатель на целочисленное значение
int **x;         //указатель на указатель на целочисленное значение
const int *b;    //указатель на целочисленную константу
int *const c=&i; //указатель-константа на целочисленную
переменную
```

Указатель типа void применяется в тех случаях, когда конкретный тип объекта, адрес которого нужно хранить, не определен.

Перед использованием указателя надо выполнить его *инициализацию*, т.е. присвоение начального значения. Существуют следующие способы инициализации указателя:

1)присваивание указателю адреса существующего объекта:

- с помощью операции получения адреса:

```
int a=50;           //целая переменная
int *x=&a;           //указателю присваивается адрес целой переменной a
int *y (&a);         // указателю присваивается адрес целой переменной a
```

- с помощью значения другого инициализированного указателя

```
int *z=x;           //указателю присваивается адрес, хранящийся в x.
```

2)присваивание указателю адреса области памяти в явном виде:

```
int *p=(int *) 0xB8000000;
```

где 0xB8000000 - шестнадцатеричная константа, *(int *)* - операция явного приведения типа к типу указатель на целочисленное значение.

3)присваивание пустого значения:

```
int *x=NULL; int *y=0;
```

где NULL стандартная константа, определенная как указатель равный 0

4) выделение участка динамической памяти и присваивание ее адреса указателю:

```
int *a = new int;      //1  
int *b = new int (50); //2
```

// 1 операция *new* выполняет выделение достаточного для размещения величины типа *int* участка динамической памяти и записывает адрес начала этого участка в переменную *a*.

Память под переменную *a* выделяется на этапе компиляции.

//2, кроме действий описанных выше, производится инициализация выделенной динамической памяти значением 50. Можно и так:

```
int *arr = new int[size];
```

Освобождение памяти, выделенной с помощью операции *new*, должно выполняться с помощью операции *delete*.

При этом переменная-указатель сохраняется и может инициализироваться повторно.

пример использования операции *delete*:

```
delete a; delete []b; delete[] arr;
```

Ссылки

Ссылка представляет собой синоним имени, указанного при инициализации ссылки.

Ссылку можно рассматривать как указатель, который разыменовывается неявным образом.

Формат объявления ссылки: **<базовый тип> & <имя ссылки>**

Например:

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{int a=50;    //целочисленная переменная a
```

```
int &b=a;    //ссылка на целочисленную переменную a
```

```
 //(ссылка b - альтернативное имя для переменной a)
```

```
#include <iostream>
using namespace std;
int main()
{int a=50;
int &b=a;
cout << a << endl << &b;
return 0;
}
```

 C:\Users\lenovo\Desktop\zzz.exe

50

0x6ffe14

Process exited after 0.1749 seconds with return value 0

Для продолжения нажмите любую клавишу . . .

2. Фактические и формальные параметры функции

Функция - это совокупность объявлений и операторов, предназначенная для решения определенной задачи.

Любая программа на C++ состоит из функций, одна из которых должна иметь имя `main` (с нее начинается выполнение программы).

Функция начинает выполняться в момент **вызова**. Любая функция должна быть *объявлена* и *определена*.

Объявление функции должно находиться в тексте раньше ее вызова для того, чтобы компилятор мог осуществить проверку правильности вызова.

В общем виде структура программы на С++ может иметь вид:

директивы компилятора

тип имя_1 (список_переменных)

{

тело_функции_1;

}

тип имя_2 (список_переменных)

{

тело_функции_2;

}

...

int main (список_переменных)

{

// тело функции main может содержать операторы вызова
функций имя_1, имя_2....

тело_основной_функции;

}

Объявление функции (прототип, заголовок, сигнатура) задает ее имя, тип возвращаемого значения и список передаваемых параметров.

Определение функции содержит, кроме объявления, *тело* функции, представляющее собой последовательность операторов и описаний в фигурных скобках:

```
[ класс ] тип имя ([ список_параметров ])[throw (
исключения )]{ тело функции }
```

Прототипом функции в языке

Си или С++ называется объявление функции, не содержащее тела функции, но указывающее имя функции, арность, типы аргументов и возвращаемый тип данных. В то время как определение функции описывает, что именно делает функция, прототип функции может восприниматься как описание её интерфейса. В прототипе имена аргументов являются необязательными, тем не менее, необходимо указывать тип вместе со всеми модификаторами (например, указатель ли это или константный аргумент).

Рассмотрим следующий прототип функции:

```
int go(int n);
```

Этот прототип объявляет функцию с именем «go», которая принимает один аргумент «n» целого типа и возвращает целое число. Определение функции может располагаться где угодно в программе, но объявление требуется только в случае её использования.

Если функция предварительно не была объявлена, а её имя встречается в выражении, сразу за которым следует открывающая скобка, то она неявно объявляется как функция, возвращающая результат типа `int` и ничего не предполагается о её аргументах. В этом случае компилятор не сможет выполнить проверку типов аргументов и аргументность, когда функция вызывается с некоторыми аргументами. Это потенциальный источник проблем.

Арность — количество аргументов или операндов (унарный - один аргумент, бинарный — два, тернарный — три, мультиарный — любое количество операндов или параметров).

Следующий код иллюстрирует ситуацию, в которой поведение неявно объявленной функции не определено.

#include <stdio.h> /* * При реализации этого прототипа компилятор выдаст сообщение об ошибке * в main(). Если он будет пропущен, то и сообщения об ошибке не будет. */

int go(**int** n); /* Прототип функции */

int main(**void**) /* Вызов функции */

{ printf("%d\n", go()); /* ОШИБКА: у go отсутствует аргумент! */

return 0; }

int go(**int** n) /* Вызываемая функция */

{ **if** (n == 0)

return 1;

else return n * go(n - 1); }

Функция «go» ожидает аргумент целого типа, находящийся в стеке при вызове. Если прототип пропущен, компилятор не может это обработать и «go» завершит операцию на некоторых других данных стека.

Включением прототипа функции вы информируете компилятор о том, что функция «go» принимает один аргумент целого типа и вы тем самым позволяете компилятору обрабатывать подобные виды ошибок.

```
#include <stdio.h>
int go(int n);
int main()
{ printf("%d\n", go(3));
return 0; }
```

```
int go(int n)
{ if (n == 0)
return 1;
else return n * go(n - 1); }
```

Process exited after 0.1023 seconds with return value 0

Для продолжения нажмите любую клавишу . . . █

С помощью необязательного модификатора **класс** можно явно задать область видимости функции, используя ключевые слова **extern** и **static**:

extern – глобальная видимость во всех модулях программы (по умолчанию);

static – видимость только в пределах модуля, в котором определена функция.

Тип возвращаемого функцией значения может быть любым, кроме массива и функции (но может быть указателем на массив или функцию). Если функция не должна возвращать значение, указывается тип `void`.

Список параметров определяет величины, которые требуется передать в функцию при ее вызове. Элементы списка параметров разделяются запятыми.

Список формальных параметров - это последовательность объявлений формальных параметров, разделенная запятыми.

Формальные параметры - это переменные, используемые внутри тела функции и получающие значение при вызове функции путем копирования в них значений соответствующих фактических параметров.

```
int n = -25;           // глобальная переменная
int modul (int n) {    // n - формальный параметр
if(n<0) n = -1 * n;
return n;
}
```

```
int main(void) {
cout << modul(n) ;    // 25, значение глобальной
переменной n будет передано в функцию
cout << n;           // -25, но работа внутри функции
//пойдёт с локальной переменной n
return 0;
}
```

- Если функция не использует параметров, то наличие круглых скобок обязательно, а вместо списка параметров рекомендуется указать слово `void`.
- В определении, в объявлении и при вызове одной и той же функции типы и порядок следования параметров должны совпадать.
- На имена параметров ограничений по соответствию не накладывается.

- Параметры функции передаются по значению и могут рассматриваться как локальные переменные, для которых выделяется память при вызове функции и производится инициализация значениями фактических параметров. При выходе из функции значения этих переменных теряются. Поскольку передача параметров происходит по значению, в теле функции нельзя изменить значения этих переменных в вызывающей функции, являющихся фактическими параметрами.

Операции адресации и разадресации (&, *)

Для получения адреса какого – либо объекта используется операция адресации &.

```
int i;
```

```
int *ip = &i; //ip – адрес переменной i
```

Для доступа к величине по её адресу используется операция разадресации *.

```
int i = 5;
```

```
int *ip = &i;
```

```
cout << (*ip); //5
```

- Тип возвращаемого значения и типы параметров совместно определяют *тип функции*.
- Для вызова функции в простейшем случае нужно указать ее имя, за которым в круглых скобках через запятую перечисляются имена передаваемых аргументов.
- Вызов функции может находиться в любом месте программы, где по синтаксису допустимо выражение того типа, который формирует функция.

Программа, которая выводит на экран треугольник, построенный из символов «звездочка» и «пробел»:

```
#include <iostream>
using namespace std;
void fun()
{
    cout<<"* ";
}
int main ()
{
    int i, j;
    for (i=0; i<5; i++)
    {
        for (j=0; j<5-i; j++)
            fun();
        cout<<"\n";
    }
    system ("pause");
    return 0;
}
```

```
* * * * *
* * * *
* * *
* *
*
```

Для продолжения нажмите любую клавишу . . .

```
#include <iostream>
using namespace std;
void fun()
{
    cout << "* ";
}
int main()
{
    int i, j;
    for (i = 0; i < 5; i++)
    {
        for (j = 0; j < 5; j++)
            fun();
        cout << "\n";
    }
    system("pause");
    return 0;
}
```

```
* * * * *  
* * * * *  
* * * * *  
* * * * *  
* * * * *
```

Для продолжения нажмите любую клавишу . . .

Пример функции, возвращающей значение при проверке пароля:

```
#include <iostream>
#include <string>

using namespace std;
string check_pass (string password)
{
    string valid_pass = "qwerty123";
    string error_message;
    if (password == valid_pass) {
        error_message = «Доступ разрешен.»;
    } else {
        error_message = «Неверный пароль!»;
    }
    return error_message;
}

int main(){
    setlocale(LC_ALL, "Russian");
    string user_pass;
    cout << «Введите пароль: »;
    getline (cin, user_pass);
    string error_msg = check_pass (user_pass);
    cout << error_msg << endl;
    return 0;
}
```

```
#include <iostream>
#include <string>

using namespace std;
string check_pass(string password)
{
    string valid_pass = "qwerty123";
    string error_message;
    if (password == valid_pass) {
        error_message = "Доступ разрешен.";
    }
    else {
        error_message = "Неверный пароль!";
    }
    return error_message;
}

int main() {
    setlocale(LC_ALL, "Russian");
    string user_pass;
    cout << "Введите пароль : ";
    getline(cin, user_pass);
    string error_msg = check_pass(user_pass);
    cout << error_msg << endl;
    return 0;
}
```


Введите пароль : йцукен777

Неверный пароль!

C:\Users\lenovo\Desktop\Белозеров\ДВГУПС\!Дисциплины\Программирование-2 c

Нажмите любую клавишу, чтобы закрыть это окно:

Введите пароль : qwerty123

Доступ разрешен.

C:\Users\Ienovo\Desktop\Белозеров\ДВГУПС\!Дисциплины\Программирование

Нажмите любую клавишу, чтобы закрыть это окно:

Программа, которая выводит таблицу умножения на заданное число

```
#include <iostream>
using namespace std;
int func(int a, int b)
{
    return (a*b);
}
int main ()
{
    int i, j;
    cout<<"i=";
    cin>>i;
    for (j=1; j<=10; j++)
        cout<<i<<"*"<<j<<"="<<func(i, j)<<" ";
    cout<<endl; system ("pause");
    return 0;
}
```

i=5

5*1=5 5*2=10 5*3=15 5*4=20 5*5=25 5*6=30 5*7=35 5*8=40 5*9=45 5*10=50

Для продолжения нажмите любую клавишу . . . ■

Найти сумму 5-и факториалов

```
#include <iostream>
using namespace std;
int fact(int); //объявление функции
```

...

```
int main()
```

```
{
```

```
int s = 0, n, x;
```

```
for (int i = 1; i <= 5; i++)
```

```
{cout << «Введите число: »;
```

```
cin >> n;
```

```
x = fact(n);
```

```
cout << " Факториал " << n  
<<"="<<x <<endl;
```

```
s += x;
```

```
}
```

```
cout << " Сумма факториалов= " << s  
<< endl;
```

```
system("pause");
```

```
return 0;
```

```
}
```

```
//функция
```

```
int fact(int n)
```

```
{
```

```
int i, p = 1;
```

```
for (i = 1; i <= n; i++) p *= i;
```

```
return p;
```

```
}
```

Обращение к функции

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

int fact(int n)
{
    int i, p = 1;
    for (i = 1; i <= n; i++) p *= i;
    return p;
}

int main()
{
    setlocale(LC_ALL, "Russian");
    int s = 0, n, x;
    for (int i = 1; i <= 5; i++){
        cout << " Введите число : ";
        cin >> n;
        x = fact(n);
        cout << " Факториал " << n << "=" << x << endl;
        s += x;    }
    cout << " Сумма факториалов= " << s << endl;
    system("pause");
    return 0;
}
```

Введите число : 1

Факториал 1=1

Введите число : 2

Факториал 2=2

Введите число : 3

Факториал 3=6

Введите число : 4

Факториал 4=24

Введите число : 5

Факториал 5=120

Сумма факториалов= 153

Для продолжения нажмите любую клавишу . . . ■

Найти число сочетаний из n по k

$$C_n^k = \frac{n!}{k! (n - k)!}.$$

```
#include <iostream>
using namespace std;
int fact (int);
...
int main ()
{
    int n, k;
    cout<<"Введите n и k ";
    cin>>n>>k;
    cout<<fact(n)/(fact(k)*fact(n-
k))<<endl;
```

```
system("pause");
return 0;
}
```

//подпрограмма

```
int fact (int n)
{
    int i, p=1;
    for (i=1;i<=n;i++) p*=i;
    return p;
```

```
}
```

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

//подпрограмма
int fact(int n)
{
    int i, p = 1;
    for (i = 1; i <= n; i++) p *= i;
    return p;
}
int main()
{
    setlocale(LC_ALL, "Russian");
    int n, k;
    cout << "Введите n и k ";
    cin >> n >> k;
    cout << fact(n) / (fact(k) * fact(n - k)) << endl;
    system("pause");
    return 0;
}
```


Введите n и k 5 2

10

Для продолжения нажмите любую клавишу . . . ■

3. Основные методы сортировки

- Сортировка методом вставок.
- Обменная сортировка.
- Сортировка посредством выбора.
- Сортировка путем подсчета.
- Специальная сортировка.

Сортировка методом вставок

Элементы просматриваются по одному, и каждый новый элемент вставляется в подходящее место среди ранее упорядоченных элементов.

шаг	отсортированная часть массива							тек. элемент	вставка
1	3							3	false
2	3	3						7	false
3	3	3	7					1	true
4	1	3	3	7				2	true
5	1	2	3	3	7			5	true
6	1	2	3	3	5	7		0	true
—	0	1	2	3	3	5	7	—	—

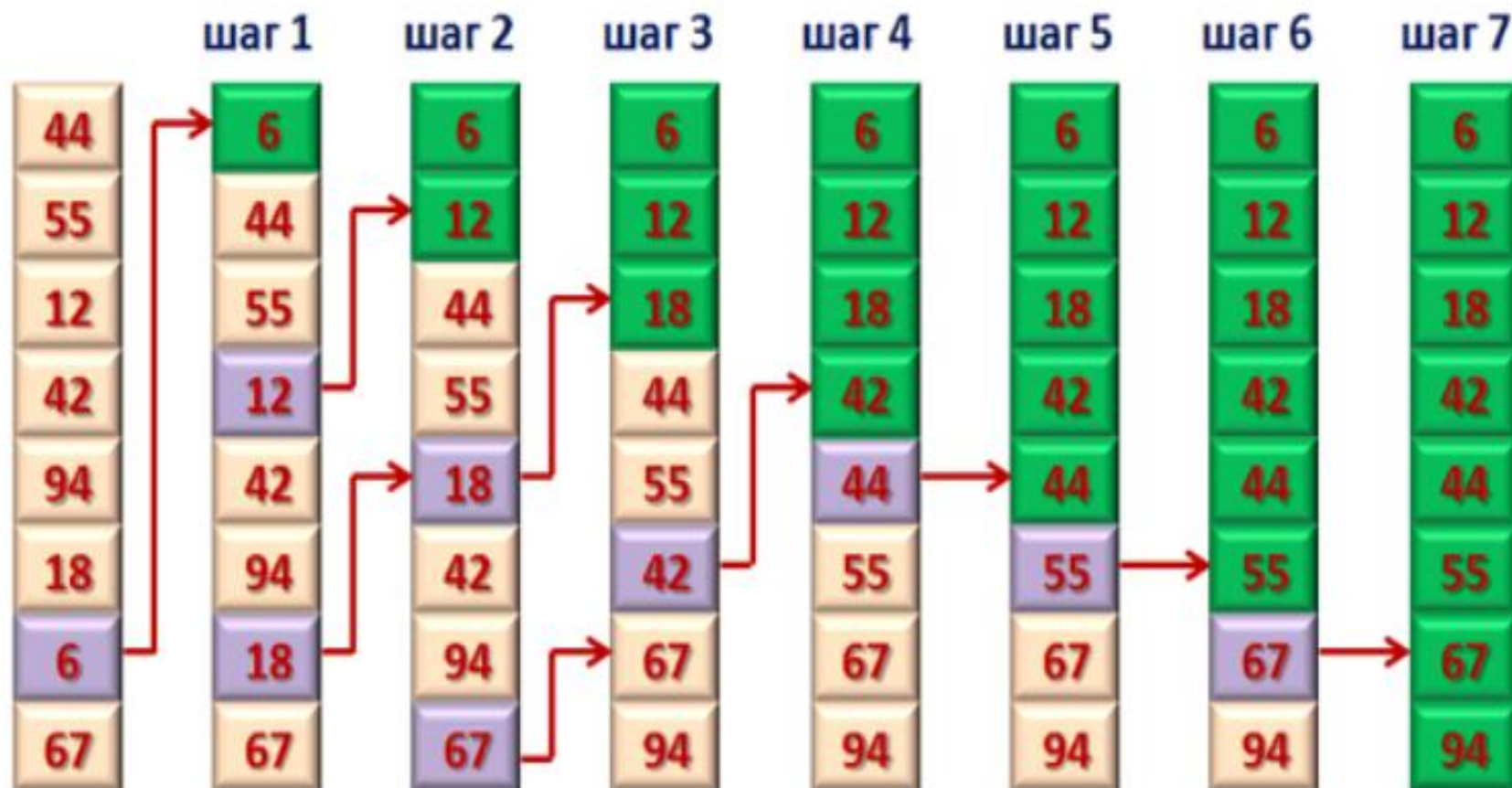
Сортируемый массив можно разделить на две части — отсортированная часть и неотсортированная. В начале сортировки первый элемент массива считается отсортированным, все остальные — не отсортированные. Начиная со второго элемента массива и заканчивая последним, алгоритм вставляет неотсортированный элемент массива в нужную позицию в отсортированной части массива. Таким образом, за один шаг сортировки отсортированная часть массива увеличивается на один элемент, а неотсортированная часть массива уменьшается на один элемент.

Рассмотрим пример сортировки по возрастанию массива из 7 чисел (см. таблицу): исходный массив: 3 3 7 1 2 5 0

Обменная сортировка

Если два элемента расположены не по порядку, то они меняются местами.

Этот процесс повторяется до тех пор, пока элементы не будут упорядочены.



```
void bubbleSort(int *num, int size)
{
    // Для всех элементов
    for (int i = 0; i < size - 1; i++)
    {
        for (int j = (size - 1); j > i; j--) // для всех
        //элементов после i-ого
        {
            if (num[j - 1] > num[j]) // если текущий элемент
            //меньше предыдущего
            {
                int temp = num[j - 1]; // меняем их местами
                num[j - 1] = num[j];
                num[j] = temp;
            }
        }
    }
}
```

Сортировка посредством выбора

Сначала выделяется наименьший (или, быть может, наибольший) элемент и каким-либо образом отделяется от остальных, затем выбирается наименьший (наибольший) из оставшихся и т.д.

9	3	7	5	1
---	---	---	---	---

исходная последовательность в массиве

1	3	7	5	9
---	---	---	---	---

шаг 0: 1 <-----> 9

1	3	7	5	9
---	---	---	---	---

шаг 1: 3 <-----> 3

1	3	5	7	9
---	---	---	---	---

шаг 2: 5 <-----> 7

1	3	5	7	9
---	---	---	---	---

шаг 3: 7 <-----> 7

1	3	5	7	9
---	---	---	---	---

массив отсортирован

```
#include <iostream>
using namespace std;
int main()
{
    const int N = 10;
    int a[N] = { 1, 25, 6, 32, 43, 5, 96, 23, 4, 55 };

    int min = 0; // для записи минимального значения
    int buf = 0; // для обмена значениями

    /***** Начало сортировки *****/
    for (int i = 0; i < N; i++)
    {
        min = i; // запомним номер текущей ячейки, как ячейки
        //с минимальным значением
```

```
// в цикле найдем реальный номер ячейки с минимальным значением
```

```
for (int j = i + 1; j < N; j++)
```

```
min = (a[j] < a[min]) ? j : min;
```

```
// сделаем перестановку этого элемента, поменяв его  
//местами с текущим
```

```
if (i != min)
```

```
{
```

```
buf = a[i];
```

```
a[i] = a[min];
```

```
a[min] = buf;
```

```
}}
```

```
/****** Конец сортировки *****/
```

```
for (int i = 0; i < N; i++) //Вывод отсортированного
```

```
//массива
```

```
cout << a[i] << '\t';
```

```
cout << endl;
```

```
}
```

1 4 5 6 23 25 32 43 55 96

C:\Users\lenovo\Desktop\Белозеров\ДВГУПС\!Дисциплины\Программирование-2 семестр\T
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "C
Нажмите любую клавишу, чтобы закрыть это окно:

Сортировка путем подсчета

Каждый элемент сравнивается со всеми остальными; окончательное положение элемента определяется после подсчета числа меньших ключей.

Специальная сортировка

Она хороша для определённого набора элементов, но не поддается простому обобщению, если элементов больше.

СПАСИБО ЗА ВНИМАНИЕ!