# Лекция №4. Использование функций при программировании на C/C++

## *4.1. Общие сведения о функциях*

Подпрограмма именованная, логически законченная группа операторов языка, которую можно вызвать для выполнения любое количество раз из различных мест программы. В языке C/C++ подпрограммы реализованы в виде функций. Функция принимает параметры и возвращает единственное скалярное значение.

```
Заголовок_функции
{
тело_функции
}
```

Заголовок функции имеет вид

**type имя_функции ([список параметров])**

*type* тип возвращаемого функцией значения;

*список параметров* список передаваемых в функцию величин, которые отделяются запятыми, каждому параметру должен предшествовать его тип;

В случае, если вызываемые функции идут до функции main, структура программы будет такой.

```
директивы компилятора
...
Тип_результата f1(Список_переменных)
{
Операторы
}
Тип_результата f2(Список_переменных)
{
Операторы
}
...
Тип_результата fn(Список_переменных)
{
Операторы
}
int main(Список_переменных)
{
Операторы основной функции, среди которых могут операторы
 вызова функций f1, f2, ..., fn
}
```

В случае, если вызываемые функции идут после функции *main*, структура программы будет такой (заголовки функций должны быть описаны до функции *main()*). Опережающие заголовки функций называют *прототипами* функций.

```
директивы компилятора
...
Тип_результата f1(Список_переменных);
Тип_результата f2(Список_переменных);
...
Тип_результата fn(Список_переменных);
int main(Список_переменных)
```

**{**
**Операторы основной функции, среди которых могут операторы**
**вызова функций f1, f2, ..., fn**
**}**
**Тип_результата f1(Список_переменных)**
**{**
**Операторы**
**}**
**Тип_результата f2(Список_переменных)**
**{**
**Операторы**
**}**
**...**
**Тип_результата fn(Список_переменных)**
**{**
**Операторы**
**}**
Для того, чтобы функция вернула какое-либо значение, в ней должен быть оператор
**return значение;**
Для вызова функции необходимо указать имя функции и в круглых скобках список передаваемых в функцию значений.

## *4.2. Передача параметров в C/C++*

Параметры, указанные в заголовке функции, называются формальными. Параметры, передаваемые в функцию, называются фактическими.
При обращении к функции фактические параметры передают свое значение формальным и больше не изменяются. Типы, количество и порядок следования формальных и фактических параметров должны совпадать. С помощью оператора *return* из функции возвращается единственное значение.
Для того чтобы функция возвращала не только скалярное значение, то в качестве передаваемого в функцию значения можно использовать указатель.
Рассмотрим все выше изложенные теоретические положения на примере решения практических задач.
ЗАДАЧА 4.1. Вводится последовательность целых чисел, 0 конец последовательности. Найти минимальное среди простых чисел и максимальное, среди чисел, не являющихся простыми.
Целое число называется простым, если оно делится нацело только на самого себя и единицу. Напомним, что алгоритм проверки, что число N является простым состоит в следующем: если разделим N без остатка хотя бы на одно число в диапазоне от 2 до N/2, то число не является простым. Если не найдем ни одного делителя числа, число N простое .Проверку является ли число N простым оформим в виде отдельной функции с именем *prostoe*. Входным параметром функции будет целое число N, функция будет возвращать значение 1, если число простое и 0 в противном случае .

```
#include <iostream>
using namespace std;
int prostoe(int N)
{
      int i,pr;
      if (N<1) pr=0;
            else
for(pr=1,i=2;i<=N/2;i++)
if (N%i==0) {pr=0;break;}
return pr;
```

```
}
int main(int argc, char* argv[])
{
      int kp=0,knp=0,min,max,N;
for (cout << "N=", cin>>N; N!=0; cout<<"N=", cin>>N)
            if (prostoe(N))
            {
                  kp++;
                  if (kp==1) min=N;
                  else if (N<min) min=N;
            }
            else
            {
                  knp++;
                  if (knp==1) max=N;
                  else if (N>max) max=N;
            }
      if (kp>0) cout <<"min= "<<min<<"\t";
      else cout <<"Net prostih";
      if (knp>0) cout <<"max="<<max<<endl;
      else cout <<"Net ne prostih";
      return 0;
}
```

ЗАДАЧА 4.2. Вводится последовательность из N целых чисел, найти среднее арифметическое совершенных чисел и среднее геометрическое простых чисел.

В этой программе кроме простых чисел будут фигурировать совершенные. Число называется совершенным, если сумма всех делителей, меньших его самого равна самому числу.

При решении этой задачи понадобятся две функции:

- функция *prostoe*,
- функция *soversh*, которая определяет является ли число совершенным; входным параметром функции будет целое число N, функция будет возвращать значение 1, если число совершенным и 0 в противном случае .

```
#include <iostream>
#include <math.h>
using namespace std;
int prostoe(int N)
{
int i,pr;
      if (N<1) pr=0;
            else
for(pr=1,i=2;i<=N/2;i++)
if (N%i==0) {pr=0;break;}
return pr;
}
int soversh(int N)
{
      int i,S;
      if (N<1) return 0;
            else
for(S=0,i=1;i<=N/2;i++)
if (N%i==0) S+=i;
if (S==N) return 1;
else return 0;
}
int main(int argc, char* argv[])
{
      int i,N,X,S,kp,ks;
```

```
      long int P;
      cout <<"N=";
      cin>>N;
      for(kp=ks=S=0,P=1,i=1;i<=N;i++)
      {
            cout <<"X=";
            cin >> X;
            if (prostoe(X))
            {
                  kp++;P*=X;
            }
            if (soversh(X))
            {
                  ks++;S+=X;
            }
      }
      if (kp>0)
cout<<"SG="<<pow(P,(float)1/kp) <<endl;
      else
cout<<"Net prostih";
      if (ks>0)
 cout <<"SA="<<(float)S/ks<<endl;
      else cout<<"Net soversh";
      }
```

Рассмотрим уже известную задачу, но решим ее с использованием функций.

ЗАДАЧА 4.3. Дано натуральное число N. Определить самую большую цифру и ее позицию в числе (N=573863, наибольшей является цифра 8, ее позиция четвертая слева ).

Первым этапом решения этой задачи будет нахождение количества разрядов в числе, для нахождения можно составить функцию kol_raz, блок-схема которой представлена на рис. 4.1

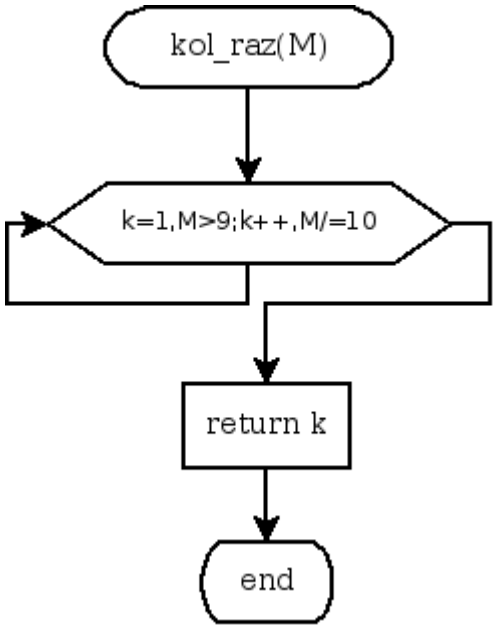


*Рисунок 4.1: Функция определения количества разрядов в числе*

Блок-схема основного алгоритма представлена на рис. 4.2.

```
#include <stdio.h>
#include <math.h>
int kol_raz(int M)
{
      int k=1;
      while(M>9)
```

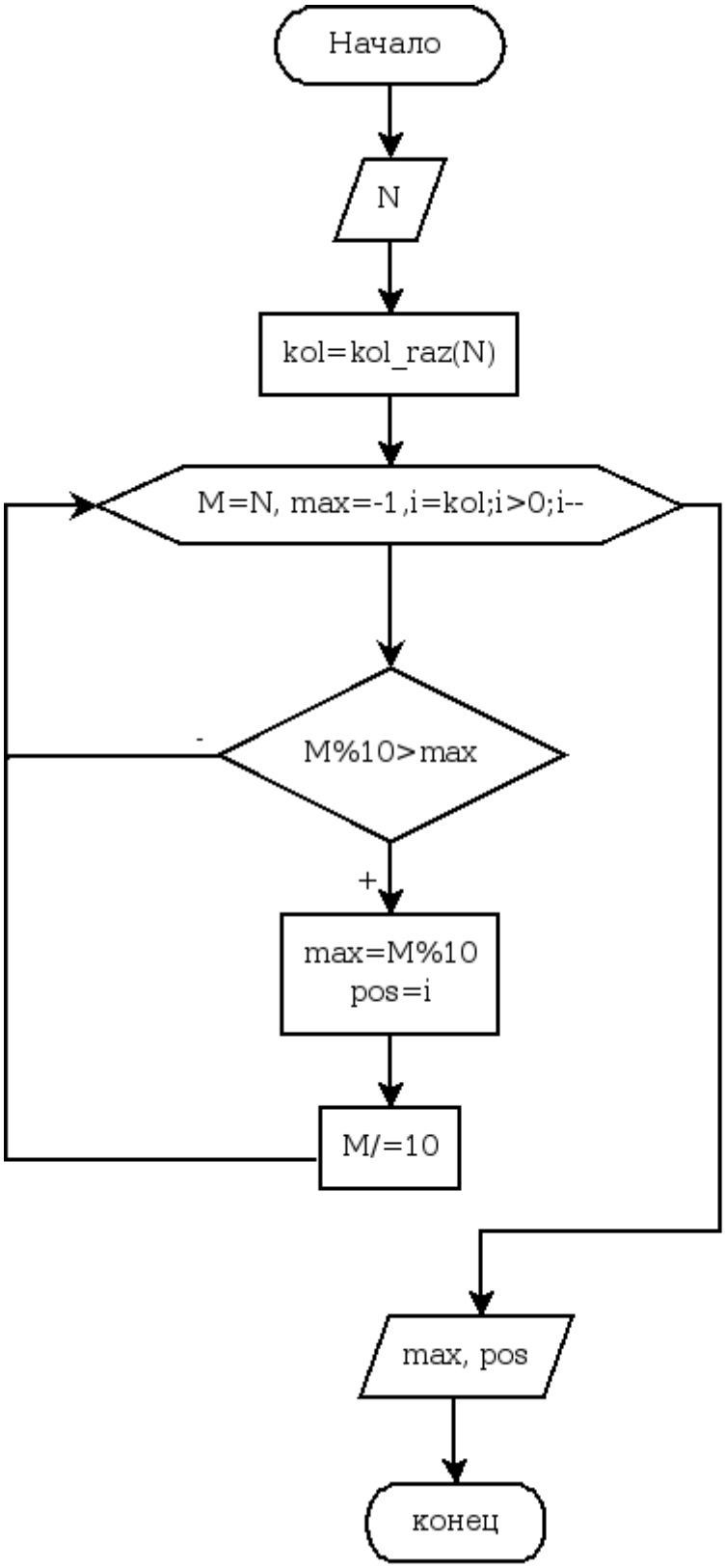


*Рисунок 4.2: Блок-схема решения задачи 4.3*

```
	{
		k++;
		M/=10;
	}
	return k;
}
int main()
```

```
{
long int N,M,kol=1;
int max,pos,i;
printf("\n N=");
// Ввод числа N.
scanf("%ld",&N);
// Вычисление количества позиций в числе (kol).
kol=kol_raz(N);
printf("V chisle %ld - %ld razryadov\n",N,kol);
// Вычисление максимальной цифры в числе, и ее номера.
for(M=N, max=-1, i=kol;i>0;i--)
{
if (M%10>max)
{
max=M%10;
pos=i;
}
M/=10;
}
// Вывод на экран максимальной цифры в числе, и ее номера.
printf("V chisle %ld maximalnaya tsifra %d, ee nomer %d\n",N,max,pos);
}
```

## *4.3. Рекурсивные функции в C/C++*

Под рекурсией в программировании понимается вызов функции из тела ее самой. В рекурсивных алгоритмах функция вызывает саму себя до выполнения какого-то условия.
Рассмотрим реализацию несколько хорошо известных алгоритмов с помощью рекурсивных функций.
1. Функция **long int factoial(int n)** предназначена для вычисления факториала числа n.

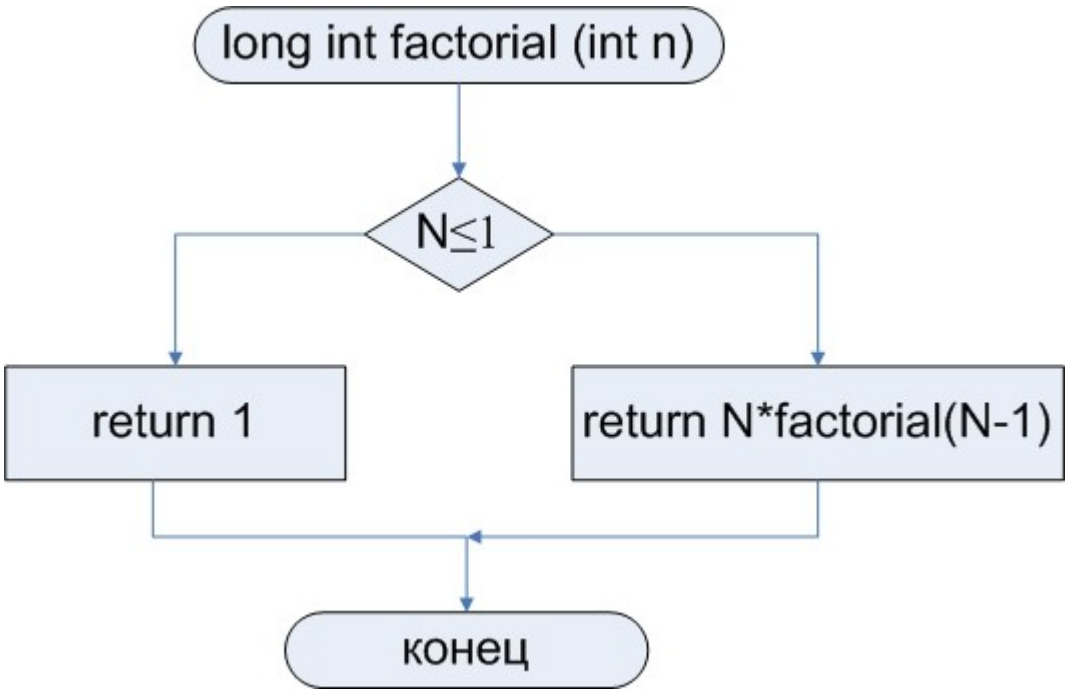


*Рисунок 4.3: Блок-схема функции вычисления факториала*

```
#include <iostream>
using namespace std;
long int factorial(int n)
{
      if (n<=1) return n;
```

```
      else
            return n*factorial(n-1);
}
int main()
{
      int i;
      long int f;
      cout<<"i=";
      cin>>i;
      f=factorial(i);
      cout<<i<<"!="<<f<<endl;
      return 0;
}
```

2. Функция **float stepen(float a, int n)** предназначена для возведения числа a в целую степень n.

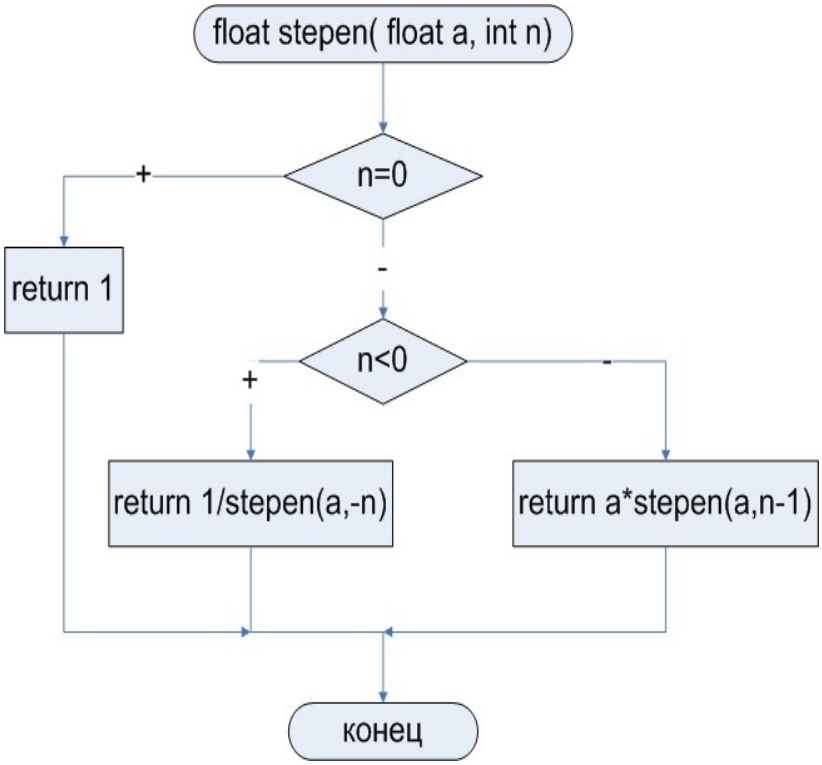


*Рисунок 4.4: Блок-схема рекурсивной функции возведения числа в целую степень*

```
#include <iostream>
using namespace std;
float stepen(float a, int n)
{
      if (n==0) return(1);
      else
            if (n<0) return(1/stepen(a,-n));
            else
                  return(a*stepen(a,n-1));
}
int main()
{
      int i;
      float s,b;
      long int f;
      cout<<"b=";
```

```
      cin>>b;
      cout<<"i=";
      cin>>i;
      s=stepen(b,i);
      cout<<"s="<<s<<endl;
      return 0;
}
```

3. Функция **long int fibonachi(int n)** предназначена для вычисления n-го числа Фибоначчи. Если нулевой элемент последовательности равен 0, первый 1, а каждый последующий равен сумме двух предыдущих, то это последовательность чисел Фибоначчи (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ... ).

```
#include <iostream>
using namespace std;
long int fibonachi(unsigned int n)
{
      if ((n==0)||(n==1)) return(n);
      else
return(fibonachi(n-1)+fibonachi(n-2));
}
int main(int argc, char* argv[])
{
      int i;
      long int f;
      cout<<"i=";
      cin>>i;
      f=fibonachi(i);
      cout<<"f="<<f<<endl;
      return 0;
}
```

## *4.4. Область видимости переменных в функциях C/C++, расширение области видимости переменных*

По месту объявления переменные в языке Си можно разделить на три класса:

- *локальные* переменные ,которые объявляются внутри функции и доступны только в ней.

Например:

```
int main()
{
float s;
s=4.5;
}
int f1()
{
int s;
s=6;
}
int f2()
{
long int s;
s=25;
}
```

В функции *main* определена вещественная переменная *s* (типа *float*), ей присвоено значение 4.5, в функции *f1* есть другая переменная *s* (типа *int*), ей присвоено значение 6, а в функции *f2* есть еще одна переменная *s* (типа *long int*), ей присвоено значение 25.

- *глобальные* переменные ,которые описаны до всех функций ,они доступны из любой

функции.

Например:

```
#include <stdio.h>
float s;
int main()
{
s=4.5;
}
int f1()
{
s=6;
}
int f2()
{
s=25;
}
```

Определена глобальная переменная *s* (типа *float*), в функции *main* ей присваивается значение 4.5, в функции *f1* присваивается значение 6,а в функции *f2* присваивается значение 25.

- *формальные* параметры функций описываются в списке параметров функции.

Рассмотрим особенности использования локальных и глобальных переменных в программах на C++:

1. Область видимости и использования локальной переменной ограничена функцией, где она определена.
2. Глобальные переменные объявляются вне любых функций и их областью видимостью является весь файл.
3. Одно и тоже имя может использоваться при определении глобальной и локальной переменной. В этом случае в функции, где определена локальная переменная действует локальное описание, вне этой функции «работает» глобальное описание. Из функции, где действует локальное описание переменной можно обратиться к глобальной переменной с таким же именем, используя оператор расширения области видимости **::переменная**.

Рассмотрим это на примере

```
#include <iostream>
using namespace std;
float pr=100.678;
int prostoe (int n)
{     int pr=1,i;
      if (n<0) pr=0;
      else
            for (i=2;i<=n/2;i++)
            if (n%i==0){pr=0;break;}
// Вывод локальной переменной
      cout<<"local pr="<<pr<<endl;
// Вывод глобальной переменной
      cout<<"global pr="<<::pr<<endl;
      return pr;
}
int main()
{
      int g;
      cout<<"g=";
      cin>>g;
      if (prostoe(g)) cout<<"g – prostoe";
else cout<<"g – ne prostoe";
      return 0;
}
```

Результаты работы программы

**g=7**
**local pr=1**
**global pr=100.678**
**g - prostoe**
Press any key to continue

## *4.5. Перегрузка и шаблоны функций*

Язык C++ позволяет связать с одним и тем же именем функции различные определения, т. е. возможно существование нескольких функций с одним и тем же именем. У этих функций может быть разное количество параметров или разные типы параметров. Создание двух или более функций с одним и тем же именем называется перегрузкой имени функции. Перегруженные функции следует создавать, когда одно и то же действие следует выполнить над разными типами входных данных, а иногда одна и также функция над разными типами входных данных выполняется с помощью разных алгоритмов.
Функция возведения в степень не определена при $0^0$ и при возведении отрицательного x в дробную степень n=k/m в случае четного m. Попробуем с использованием механизма перегрузки написать более универсальную, чем функция pow (из библиотеки math.h) функцию Pow. Наша функция в случаях неопределенности операции возведения в степень будет возвращать 0.

```
#include <iostream>
#include <math.h>
using namespace std;
float Pow(float a, int k, int m)
//Функция возведения вещественного числа в степень k/m.
{
      cout<<"function 1\t";
      if (a==0) return (0);
      else
            if (k==0) return(1);
            else
            if (a>0)
             return(exp((float)k/m*log(a)));
            else
                  if (m%2!=0)
      return (-(exp((float)k/m*log(-a))));
}
float Pow(float a, int n)
//Функция возведения вещественного числа в целую степень n.
{
      if (a==0)
{cout<<"function 2\t";return (0);}
      else
      if (n==0)
{cout<<"function 2\t";return (1);}
      else
            if (n<0) return(1/pow(a,-n));
            else
                  return(a*pow(a,n-1));
}
int Pow(int a, int n)
//Функция возведения целого числа в целую степень n.
{
      if (a==0)
 {cout<<"function 3\t";return (0);}
```

```
      else
      if (n==0)
{cout<<"function 3\t";return (1);}
      else
            if (n<0) return(1/pow(a,-n));
            else
                  return(a*pow(a,n-1));
}
int main()
{
      float a;
      int k,n,m;
      cout<<"a=";
      cin>>a;
      cout<<"k=";
      cin>>k;
      cout<<"s="<<Pow(a,k)<<endl;
      cout<<"s="<<Pow((int)a,k)<<endl;
      cout<<"a=";
      cin>>a;
      cout<<"k=";
      cin>>k;
      cout<<"m=";
      cin>>m;
      cout<<"s="<<Pow(a,k,m)<<endl;
      return 0;
}
```

Результаты работы программы

**a=5.2**
**k=3**
**function 2 s=140.608**
**function 3 s=125**
**a=-8**
**k=1**
**m=3**
**function 1 s=-2**
**Press any key to continue**

Если перегрузку можно применять при использовании различных алгоритмов решения задачи при различных типах исходных данных и просто при различных типах исходных данных, то при использовании различных типов исходных данных можно применять шаблоны.

*Шаблон* это особый вид функций , который начинается со служебного слова template, за которым в угловых скобках (<>) следует список используемых в функции типов данных. Каждый тип предваряется служебным словом class. Можно сказать, что в случае шаблона в качестве параметров выступают не только переменные, но их типы.

Рассмотрим пример шаблона поиска наименьшего из четырех чисел.

```
#include "stdafx.h"
#include <iostream.h>
//Определяем абстрактный тип данных с помощью служебного слова Type
template <class Type>
// Определяем функцию поиска минимума с использованием типа данных Type
Type minimum(Type a, Type b, Type c, Type d)
{
      Type min=a;
      if (b<min) min=b;
      if (c<min) min=c;
      if (d<min) min=d;
```

```
      return min;
}
int main()
{
      int ia,ib,ic,id,mini;
      float ra,rb,rc,rd,minr;
      cout<<"Введите 4 целых числа\t";
      cin>>ia>>ib>>ic>>id;
//Вызов функции minimum, в которую передаем 4 целых значениями
      mini=minimum(ia,ib,ic,id);
      cout<<"\n"<<mini<<"\n";
      cout<<"Введите 4 вещественных числа\t\t";
      cin>>ra>>rb>>rc>>rd;
//Вызов функции minimum, в которую передаем 4 вещественных значениями
      minr=minimum(ra,rb,rc,rd);
      cout<<"\n"<<minr<<"\n";
      return 0;
}
```

## 4.6. Использование значений формальных параметров по умолчанию

В C++ существует возможность задать значение некоторых формальных параметров по умолчанию, если такой параметр будет отсутствовать в вызове функции, то будет работать значение по умолчанию. Формальные параметры со значениями по умолчанию должны быть самыми последними в списке. В качестве примера рассмотрим функцию возведения вещественного числа a в целую степень n. Параметр n имеет значение по умолчанию равное 3, если при вызове функции n будет опущено, то число будет возводится в степень 3.

```
#include <iostream.h>
using namespace std;
float stepen(float a, int n=3)
{
      if (n==0) return(1);
      else
            if (n<0) return(1/stepen(a,-n));
            else
                  return(a*stepen(a,n-1));
}
int main()
{
      int a;
      long int f;
      cout<<"a=";
      cin>>a;
//Возведение числа a в пятую степень.
      f=stepen(a,5);
      cout<<"f="<<f<<endl;
//Возведение числа a в третью степень.
      f=stepen(a);
      cout<<"f="<<f<<endl;
      return 0;
}
```