5. Использование функций при программировании на C/C++

5.1 Общие сведения о функциях. Локальные и глобальные переменные

Функция – это поименованный набор описаний и операторов, выполняющих определенную задачу. Функция может принимать параметры и возвращать значение. Информация, передаваемая в функцию для обработки, называется *параметром*, а результат вычислений функции ее *значением*. Обращение к функции называют *вызовом*. Как известно любая программа на С состоит из одной или нескольких функций. При запуске программы первой выполняется функция `main`. Если среди операторов функции `main` встречается вызов функции, то управление передается операторам функции. Когда все операторы функции будут выполнены, управление возвращается оператору, следующему за вызовом функции.

Перед вызовом функция должны быть обязательно описана.

Описание функции состоит из заголовка и тела функции:

```
тип имя_функции(список_переменных)
{
     тело_функции
}
```

Заголовок функции содержит:

- `тип`, возвращаемого функцией значения, он может быть любым; если функция не возвращает значения, указывают тип `void`;
- `имя_функции`;
- `список_переменных` – перечень передаваемых в функцию величин (*аргументов*), которые отделяются друг от друга запятыми; для каждой переменной из списка указывается `тип` и `имя`; если функция не имеет аргументов, то в скобках указывают либо тип `void`, либо ничего.

Тело функции представляет собой последовательность описаний и операторов, заключенных в фигурные скобки.

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;
}
...
тип имя_n(список_переменных)
{
     тело_функции_n;
}
int main(список_переменных)
{
//Тело функции может содержать операторы вызова
//функций имя_1, имя_2, ..., имя_n
     тело_основной_функции;
}
```

Однако допустима и другая *форма записи программного кода*:

```
директивы компилятора
тип имя_1(список_переменных);
тип имя_2(список_переменных);
...
тип имя_n(список_переменных);
int main(Список_переменных)
{
//Тело функции может содержать операторы вызова
//функций имя_1, имя_2, ..., имя_n
      тело__основной_функции;
}
тип имя_1(список_переменных)
{
     тело_функции_1;
}
тип имя_2(список_переменных)
{
     тело_функции_2;
}
...
тип имя_n(список_переменных)
{
     тело_функции_n;
}
```

Здесь функции описаны после функции `main()`, однако до нее перечислены заголовки всех функций. Такого рода опережающие заголовки называют *прототипами функций*. Прототип указывает компилятору тип данных, возвращаемых функцией, тип переменных, выступающих в роли аргументов и порядок их следования. Прототипы используются для проверки правильности вызова функций в основной программе. Имена переменных, указанные в прототипе функции, компилятор игнорирует:

```
//Записи равносильны.
int func(int a,int b);
int func(int ,int);
```

Вызвать функцию можно в любом месте программы. Для *вы-*

зова функции необходимо указать ее имя и в круглых скобках, через запятую перечислить имена или значения аргументов, если таковые имеются:

```
имя_функции(список_переменных);
```

Рассмотрим пример. Создадим функцию f(), которая не имеет входных значений и не формирует результат. При вызове этой функции на экран выводится строка символов "Счастливого Нового Года, ".

```
#include <stdio.h>
void f()        //Описание функции.
{
      printf("Счастливого Нового Года, ");
}
int main()
{
   f();         //Вызов функции.
   printf("Студенты!\n");
   f();     //Вызов функции.
   printf("Учителя!\n");
   f();     //Вызов функции.
   printf("Люди!\n");
}
```

Результатом работы программы будут три строки:

Счастливого Нового Года, Студенты!
Счастливого Нового Года, Учителя!
Счастливого Нового Года, Люди!

Далее приведен пример программы, которая пять раз выводит на экран фразу "Здравствуй Мир!". Операция вывода строки символов оформлена в виде функции fun(). Эта функция так же не имеет входных значений и не формирует результат. Вызов функции осуществляется в цикле:

```
#include <stdio.h>
void fun()
{
      printf("Здравствуй мир!\n");
}
```

```
int main()
{
   for(int i=1;i<=5;fun(),i++);
}
```

Если тип возвращаемого значения не `void`, то функция может входить в состав выражений. *Типы и порядок следования переменных в определении и при вызове функции должны совпадать*. Для того чтобы *функция вернула* какое-либо значение, в ней должен быть оператор:

`return` `выражение;`

Работает оператор следующим образом. Вычисляется значение `выражения`, указанного после `return` и преобразуется к типу возвращаемого функцией значения. Выполнение функции завершается, а вычисленное значение передается в вызывающую функцию. Любые операторы, следующие в функции за оператором `return`, игнорируются. Программа продолжает свою работу с оператора следующего за оператором вызова данной функции.

Оператор `return` может отсутствовать в функциях типа `void`, если возврат происходит перед закрывающейся фигурной скобкой, и в функции main.

Переменные, описанные внутри функции, а так же переменные из списка аргументов, являются *локальными*. Например, если программа содержит пять разных функций, в каждой из которых описана переменная N, то для С это пять различных переменных. Область действия локальной переменной не выходит за рамки функции. Значения локальных переменных между вызовами одной и той же функции не сохраняются.

Переменные, определенные до объявления всех функций и доступные всем функциям, называют *глобальными*. В функции глобальную переменную можно отличить, если не описана локальная переменная с теми же именем.

Обмен информацией между вызываемой и вызывающей функциями осуществляется с помощью *механизма передачи параметров*. `Список_переменных`, указанный в заголовке функции называется *формальными параметрами* или просто *параметрами* функции. `Список_переменных` в операторе вызова функции – это *фактические параметры* или *аргументы*.

Механизм передачи параметров обеспечивает замену формальных параметров фактическими параметрами, и позволяет выполнять функцию с различными данными. Между фактическими параметрами в операторе вызова функции и формальными параметрами в заголовке функции устанавливается взаимно однозначное соответствие. *Количество, типы и порядок следования формальных и фактических параметров должны совпадать*.

Передача параметров выполняется следующим образом. Вычисляются выражения, стоящие на месте фактических параметров. В памяти выделяется место под формальные параметры, в соответствии с их типами. Затем формальным параметрам присваиваются значения фактических. Выполняется проверка типов и при необходимости выполняется их преобразование. При несоответствии типов выдается диагностическое сообщение.

5.2 Решение задач с использованием функций

Рассмотрим несколько задач с применением функций.

ЗАДАЧА 5.1. Вводится последовательность из N целых чисел, найти среднее арифметическое совершенных чисел и среднее геометрическое простых чисел последовательности.

Для решения поставленной задачи понадобятся две функции:

•`prostoe` – определяет, является ли число простым, аргумент функции целое число `N`; функция возвращает `1`, если число простое и `0` – в противном случае.;

•`soversh` – определяет, является ли число совершенным; входной параметр целое число `N`; функция возвращает `1`, если число совершенным и `0` – в противном случае.

```
#include <stdio.h>
#include <math.h>
unsigned int prostoe(unsigned int N)
{
   int i,pr;
   for(pr=1,i=2;i<=N/2;i++)
   if (N%i==0) {pr=0;break;}
   return pr;
}
unsigned int soversh(unsigned int N)
```

```
{
  unsigned int i,S;
   for(S=0,i=1;i<=N/2;i++)
      if (N%i==0) S+=i;   //Сумма делителей.
   if (S==N) return 1;
   else return 0;
}
int main()
{
   unsigned int i,N,X,S,kp,ks;
   long int P;
//Ввод N - количества чисел
последовательности.
   printf("N="); scanf("%d",&N);
   for(kp=ks=S=0,P=1,i=1;i<=N;i++)
   {
      printf("X="); scanf("%d",&X);
      if (prostoe(X))
      {
           kp++;
           P*=X;
      }
      if (soversh(X))
      {
           ks++;
           S+=X;
      }
   }
   if (kp>0)
   printf("Среднее геометрическое простых
чисел=%8.3f\n", pow(P,(float) 1/kp));
   else    printf("Нет простых чисел\n");
   if (ks>0)
      printf("Среднее  арифметическое
совершенных чисел=%8.3f\n", (float) S/ks);
   else
      printf("Нет совершенных чисел\n");
return 0;
```

```
}
```

ЗАДАЧА 5.2. Вводится последовательность целых чисел, 0 – конец последовательности. Найти минимальное число среди простых чисел и максимальное, среди чисел, не являющихся простыми.

Текст программы:

```
#include <stdio.h>
unsigned int prostoe(unsigned int N)
{
   int i,pr;
   for(pr=1,i=2;i<=N/2;i++)
      if (N%i==0) {pr=0;break;}
   return pr;
}
int main()
{
   int kp=0,knp=0,min,max,N;
   for (printf("N="), scanf("%d",&N); N!=0;
printf("N="), scanf("%d",&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)
      printf("min= %d\t",min);
   else
      printf("Нет простых чисел.\n");
   if (knp>0)
      printf("max= %d\n",max);
```

```
   else
      printf("Нет не простых чисел\n");
return 0;
}
```

ЗАДАЧА 5.3. Вводится последовательность из N целых положительных чисел. В каждом числе найти наименьшую и наибольшую цифры[1].

Программный код к задаче 5.3 с комментариями представлен ниже.

```
#include <stdio.h>
unsigned int Cmax(unsigned long long int P)
{
unsigned int max;
if (P==0) max=0;
for (int i=1 ; P!=0;P/=10)
{
   if (i==1) {max=P%10;i++;}
   if (P%10>max) max=P%10;
}
return max;
}
unsigned int Cmin(unsigned long long int P)
{
unsigned int min;
if (P==0) min=0;
for (int i=1; P!=0;P/=10)
{
   if (i==1) {min=P%10;i++;}
   if (P%10<min) min=P%10;
}
return min;
}
int main()
{
unsigned int N, k;
unsigned long long int X;
for (printf("N="),
```

1 Выделение цифры из числа подробно описано в задаче 3.16

```
scanf("%d",&N),k=1;k<=N;k++)
{
   printf("X="); scanf("%lld",&X);
   printf("Максимальная цифра=%d",Cmax(X));
   printf("  Минимальная цифра=%d\n",
Cmin(X));
}
return 0;
}
```

ЗАДАЧА 5.4. Вводится последовательность целых положительных чисел, 0 — конец последовательности. Определить сколько в последовательности чисел-палиндромов[2].

```
#include <stdio.h>
#define uli unsigned long int
int palindrom(uli n)
{
   uli p=0,k=n;
   for(;n!=0;n/=10)
   p=p*10+n%10;

   return k==p;
}
int main(int argc, char **argv)
{
   uli z;
   unsigned int k=0;
   for(printf("z="),scanf("%li",&z);z!
=0;printf("z="),scanf("%li",&z))
   if (palindrom(z)) {k++;
   printf("Число %ld - палиндром!!!\n",z);}
   else printf("Число %ld не является
   палиндромом!!!\n",z);
   printf("Было введено %d палиндрома\n",k);
   return 0;
}
```

ЗАДАЧА 5.5. Заданы два числа - X в двоичной системе счисления, Y в системе счисления с основанием пять. Найти

2 Палиндром — любой симметричный относительно своей середины набор символов.

сумму этих чисел. Результат вывести в десятичной системе счисления.

```
#include <stdio.h>
unsigned long long int DecNC(unsigned long
long int N,unsigned int b)
{
   //Функция выполняет перевод числа N,
   //заданного в b-ичной системе счисления,
   //в десятичную систему счисления
   unsigned long long int S,P;
   //for (S=0,P=1,i=1;N/10>0;S+=N
//%10*P,P*=b,N/=10);
   for (S=0,P=1;N!=0;S+=N%10*P,P*=b,N/=10);
   return S;
}
int main()
{
unsigned long long int X,Y; unsigned int
bX,bY;
printf("X="); scanf("%lld",&X);
//Ввод числа X.
printf("bX="); scanf("%d",&bX);
//Ввод основания с/с.
printf("Y="), scanf("%lld",&Y);
//Ввод числа X.
printf("bY="); scanf("%d",&bY);
//Ввод основания с/с.
//Вывод заданных чисел в десятичной с/с.
printf("%lld(%d)=%lld(10)\
n",X,bX,DecNC(X,bX));
printf("%lld(%d)=%lld(10)\
n",Y,bY,DecNC(Y,bY));
//Вычисление суммы и вывод результата.
printf("%lld(%d)+%lld(%d)=",X,bX,Y,bY);
printf("%lld(10)\n",DecNC(X,bX)
+DecNC(Y,bY));
return 0;
```

```
}
```

ЗАДАЧА 5.6. Задано число *X* в десятичной системе счисления. Выполнить перевод числа в системы счисления с основанием 2, 5 и 7.

```
#include <stdio.h>
unsigned long long int NC(unsigned long long
int N, unsigned int b)
//Функция выполняет перевод числа N,
//заданного в десятичной системе счисления,
//в b-ричную систему счисления.
{
   unsigned long long int S,P;
   for (S=0,P=1;N!=0;S+=N%b*P,P*=10,N/=b);
   return S;
}
int main()
{
   unsigned long long int X;
   printf("X="); scanf("%lld",&X);
   printf("%lld(10)=%lld(2)\n",X,NC(X,2));
   printf("%lld(10)=%lld(5)\n",X,NC(X,5));
   printf("%lld(10)=%lld(7)\n",X,NC(X,7));
   return 0;
}
```

5.3 Рекурсивные функции

Под *рекурсией* в программировании понимают функцию, которая вызывает сама себя. *Рекурсивные функции* чаще всего используют для компактной реализации рекурсивных алгоритмов. Классическими рекурсивными алгоритмами могут быть возведение числа в целую положительную степень, вычисление факториала. С другой стороны любой рекурсивный алгоритм можно реализовать без применения рекурсий. Достоинством рекурсии является компактная запись, а недостатком расход памяти на повторные вызовы функций и передачу параметров, существует опасность переполнения памяти.

Рассмотрим применение рекурсии на примерах.

ЗАДАЧА 5.7. Вычислить факториал числа *n*.

Для решения этой задачи с применением рекурсии создадим функцию `factoial`, алгоритм которой представлен на рис.

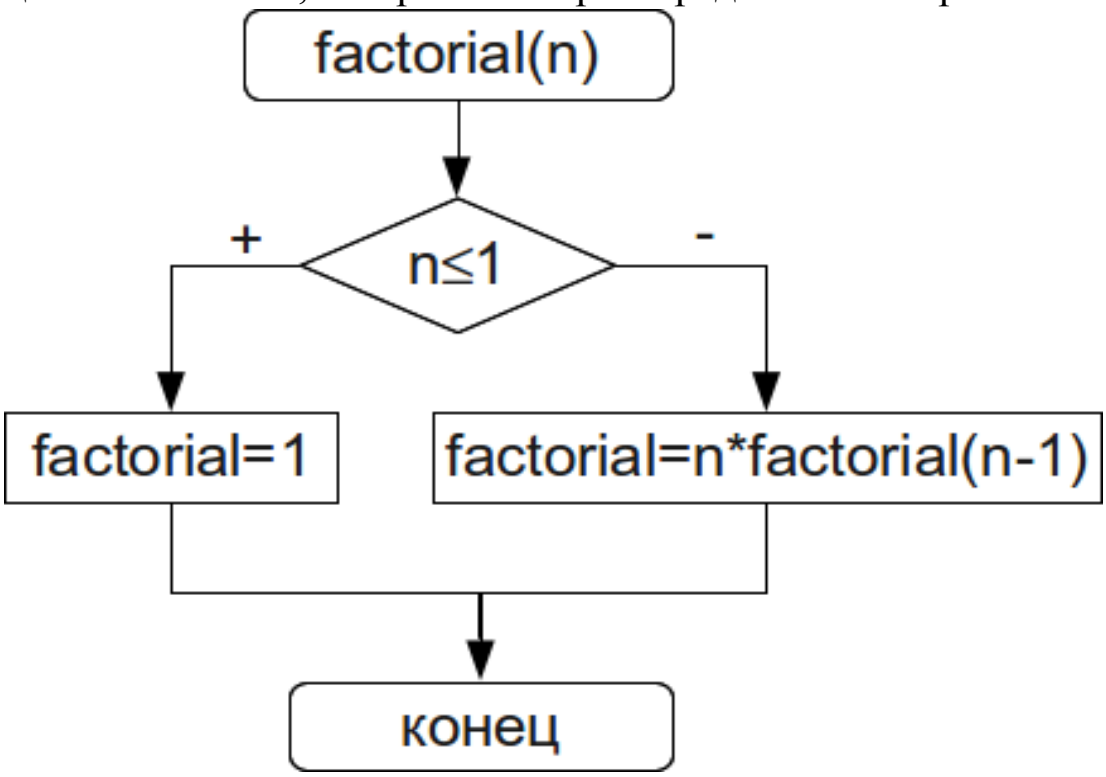

Рисунок 5.1

Текст программы с применением рекурсии:

```
#include <stdio.h>
unsigned long long int factorial(unsigned
int n)
```

```
{
   if (n<=1)
      return n;
   else
      return n*factorial(n-1);
}
int main()
{
   unsigned int i; unsigned long long int f;
   printf("i="); scanf("%d",&i);
   f=factorial(i);
   printf("%d!=%lld\n",i,f);
   return 0;
}
```

ЗАДАЧА 5.8. Вычислить *n*-ю степень числа *a* (*n* – целое число).

Результатом возведения числа *a* в целую степень *n* является умножение этого числа на себя *n* раз. Но это утверждение верно, только для положительных значений *n*. Если *n* принимает отрицательные значения, то $a^{-n}=\frac{1}{a^n}$. В случае *n*=0, $a^0=1$.

Для решения задачи создадим рекурсивную функцию `stepen`, алгоритм которой представлен на рис. 5.1.

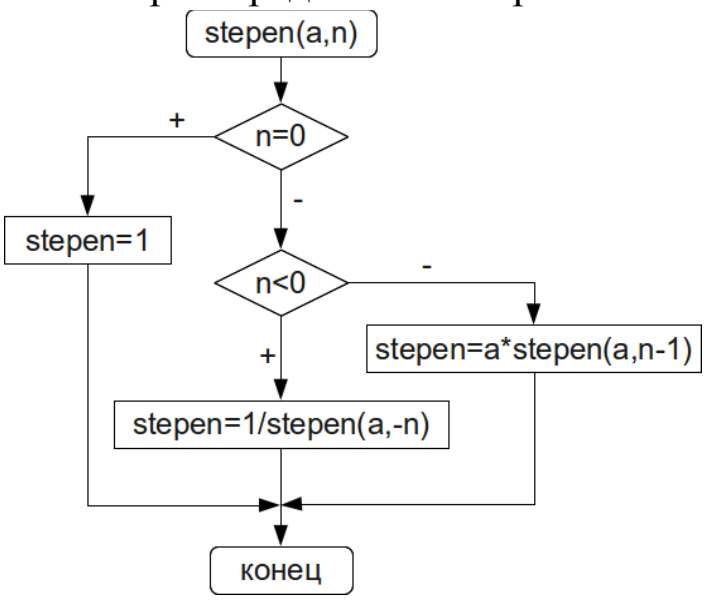

Рис. 5.1 Рекурсивный алгоритм вычисления степени числа

Текст программы с применением рекурсии:

```
#include <stdio.h>
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;
   printf("b=");scanf("%f",&b);
   printf("i=");scanf("%d",&i);
   s=stepen(b,i);
   printf("s=%8.3f\n",s);
   return 0;
}
```

ЗАДАЧА 5.9. Вычислить *n*-е число Фибоначчи.

Если нулевой элемент последовательности равен нулю, первый – единице, а каждый последующий представляет собой сумму двух предыдущих, то это *последовательность чисел Фибоначчи* (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...).

Алгоритм рекурсивной функции `fibonachi` изображен на рис. 5.2.

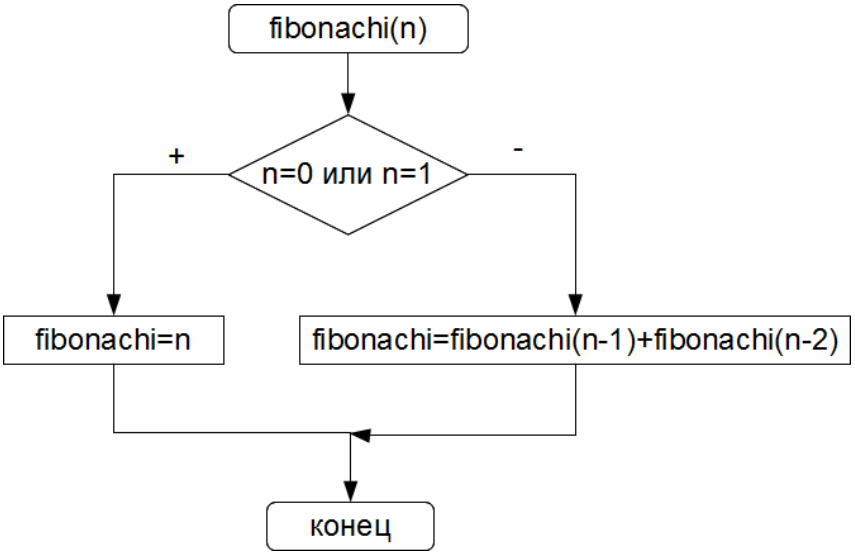

Рис. 5.2 Рекурсивный алгоритм вычисления числа Фибоначчи

Текст программы:

```
#include <stdio.h>
long int fibonachi(unsigned int n)
{
   if ((n==0)||(n==1))
      return n;
   else
      return fibonachi(n-1)+fibonachi(n-2);
}
int main()
{
   int i; long int f;
   printf("i=");scanf("%d",&i);
   f=fibonachi(i);
   printf("f=%ld\n",f);
   return 0;
}
```

5.4 Новые возможности работы с функциями в C++[3]

В языке C++ появились возможности работы с функциями, которых нет в классическом C.

3 Автор счёл нужным расширить раздел о функциях в C новыми возможностями, появившимися в C++. Это связано с тем, что эти возможности удобно использовать в классических структурных программах.

5.4.1 Перегрузка функций

Так компиляторы C++ позволяют связать с одним и тем же именем функции различные определения, то есть возможно существование нескольких функций с одинаковым 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 создают, когда одно и то же действие следует выполнить над разными типами входных данных.

В приведённом далее тексте программы три функции с именем `Pow`.

Первая выполняет операцию возведения вещественного числа a в дробную степень $n=\frac{k}{m}$, где k и m – целые числа.

Вторая возводит вещественное число a в целую степень n, третья – целое число a в целую степень n.

Какую именно функцию вызвать компилятор определяет по типу фактических параметров. Так, если a – вещественное число, а k – целое, то оператор `Pow(a,k)` вызовет вторую функцию, так как она имеет заголовок `float Pow(float a, int n)`. Команда `Pow((int)a,k)` приведет к вызову третьей функции `float Pow(int a, int n)`, так как вещественная переменная `a` преобразована к целому типу. Первая функция `float Pow(float a, int k, int m)` имеет три параметра, значит, обращение к ней осуществляется командой `Pow(a,k,m)`.

```
#include <stdio.h>
#include <math.h>
float Pow(float a, int k, int m)
   //Первая функция
{
   printf("Функция 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)
   //Вторая функция
{
   float p; int i;
   printf("Функция 2 \t");
   if (a==0)
      return 0;
   else if (n==0)
      return 1;
   else if (n<0)
   {
      n=-n;
      p=1;
      for(i=1;i<=n;i++)
      p*=a;
      return (float)1/p;
   }
   else
   {
      p=1;
      for(i=1;i<=n;i++)
      p*=a;
      return p; }

}
float Pow(int a, int n)
   //Третья функция
{
   int i,p;
   printf("Функция 3 \t");
   if (a==0)
      return 0;
   else if (n==0)
      return 1;
   else if (n<0)
```

```
    {
       n=-n;
       p=1;
       for(i=1;i<=n;i++)
       p*=a;
       return (float)1/p;
    }
    else
    {
       p=1;
       for(i=1;i<=n;i++)
       p*=a;
       return p; }

}
int main()
{
   float a; int k,m;
   printf("a=");scanf("%f",&a);
   printf("k=");scanf("%d",&k);
//Вызов 2-й функции.
   printf("s=%10.4f\n",Pow(a,k));
//Вызов 3-й функции.
   printf("s=%8.4f\n",Pow((int)a,k));
   printf("a=");scanf("%f",&a);
   printf("k=");scanf("%d",&k);
   printf("m=");scanf("%d",&m);
//Вызов 1-й функции.
   printf("s=%10.4f\n",Pow(a,k,m));
   return 0;
}
```

Результаты работы программы

```
a=5.2
k=3
Функция 2  s=140.608
Функция 3  s=125
a=-8
k=1
```

```
m=1
Функция 1   s=-8

a=5.2
k=-3
Функция 2   s=0.00711197
Функция 3   s=0.008
a=-8
k=1
m=3
Функция 1   s=-2
```

5.4.2 Шаблоны функций

Шаблон – это особый вид функции. С помощью шаблона функции можно определить алгоритм, который будет применяться к данным различных типов. Механизм работы шаблона заключается в том, что на этапе компиляции конкретный тип данных передаётся в функцию в виде параметра.

Простейшую функцию–шаблон в общем виде можно записать так:

```
template <class Type> заголовок
{
     тело функции
}
```

Обычно в угловых скобках указывают список используемых в функции типов данных. Каждый тип предваряется служебным словом `class`. В общем случае в списке могут быть не только типы данных, но и имена переменных.

Рассмотрим пример шаблона поиска наименьшего из четырех чисел.

```
#include <stdio.h>
template <class Type>
//Определяем абстрактный
//тип данных с помощью
//служебного слова Type.

Type minimum(Type a, Type b, Type c, Type d)
{ //Определяем функцию
```

```
//с использованием
//типа данных Type.
   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;
   printf("Ввод 4-х целых чисел\t");
   scanf("%d%d%d%d",&ia,&ib,&ic,&id);
   mini=minimum(ia,ib,ic,id);
//Вызов функции minimum,
//в которую передаем 4
//целых значения.
   printf("Минимум=%d\n",mini);
   printf("Ввод 4-х вещественных чисел\t");
   scanf("%f%f%f%f",&ra,&rb,&rc,&rd);
   minr=minimum(ra,rb,rc,rd);
//Вызов функции minimum,
//в которую  передаем 4
//вещественных значения.
   printf("Минимум=%6.2f\n",minr);
   return 0;
}

#include <iostream>
using namespace std;
template <class T>
T f(T x)
{
 T x2=2*x;
 return x2+(x*x+1)/x2;

}
```

```
int main(int argc, char **argv)
{
 cout<<f(5.0)<<endl<<f(5)<<endl;
 return 0;
}
```

5.4.3 Расширение области видимости переменных в функциях на языке C++

Как известно, по месту объявления переменные в языке C++ делятся на три класса: локальные, глобальные и переменные, описанные в списке формальных параметров функций. Все эти переменные имеют разную область видимости.

Локальные переменные объявляются внутри функции и доступны только в ней. О таких переменных говорят, что они имеют локальную видимость, то есть, видимы только внутри функции.

Глобальные переменные описывают вне всех функций. Поскольку они доступны из любой точки программы, то их область видимости охватывает весь файл.

Одно и тоже имя может использоваться при определении глобальной и локальной переменной. В этом случае в теле функции локальная переменная имеет преимущество и «закрывает» собой глобальную. Вне этой функции «работает» глобальное описание переменной.

Из функции, где действует локальное описание переменной можно обратиться к глобальной переменной с таким же именем, используя *оператор расширения области видимости*

```
::переменная;
```

Рассмотрим пример:

```
#include <stdio.h>
float pr=100.678; //Переменная pr определена
глобально.
int prostoe (int n)
{
int pr=1,i;
//Переменная pr определена локально.
if (n<0) pr=0;
else
     for (i=2;i<=n/2;i++)
     if (n%i==0){pr=0;break;}
printf("local pr=%d\n",pr);
//Вывод локальной переменной.
printf("global pr=%f\n",::pr);
//Вывод глобальной переменной.
return pr;
}
int main()
{
int g;
printf("g="); scanf("%d",&g);
if (prostoe(g)) printf("%d – простое \n",g);
else printf("%d – не является простым \
n",g);
return 0;
}
```

Результаты работы программы:

```
g=7
local pr=1     //Локальная переменная.
global pr=100.678001  //Глобальная
переменная.
7 – простое
```