**Функции и рекурсии**

**Функции**

Функция — это некоторый участок кода, который может многократно выполняться с различными входными значениями входных параметров.

Структура определения функции следующая:

<тип результата> <имя функции> (<список формальных параметров>)

{

<тело функции>;

[<return значения>];

}

Тип результата функции может быть любым типом С++ кроме массива.

Если функция ничего не должна возвращать, то следует указать тип результата void.

Оператор return служит для того, чтобы вернуть результат работы функции в точку вызова внутри main. После оператора return функция завершает свою работу.

Если в функции есть несколько return-ов, тогда функция завершается по достижению первого из них. Если тип результата void, то return не нужен.

Return может возвращать константное значение, переменную, выражение: return 5, return x, return 2\*x+3. Главное, чтобы тип возвращаемого значения совпадал с типом результата функции.

Список параметров — это множество пар <тип> <имя>, разделенных между собой запятыми, которое определяет количество и тип, входящих в функцию параметров.

Само по себе объявление функции не ведет ни к каким действиям. Оно только сообщает компилятору о возможности использования такой функции.

Пример.

void f (int n)

{

for (int i=0;i<n;i++)

cout<< «\*»;

cout<<endl;

}

int main ()

{

int k;

cout<< «Введите количество»<<endl;

cin>>k;

f(k);

….

f(20);

return 0;

}

Пример. Найти количество простых элементов массива.

Функция проверяет число на простоту:

bool prost (int x)

{

int del=0;

for (int i=1; i<=x;i++)

if (x%i==0)

del++;

return (del==2);

}

Нередко возникают ситуации, когда вызов функции располагается выше ее объявления или вообще в другом файле.

Для их решения необходимо использовать прототипы функции.

Прототип — это заголовок функции, то есть структура следующего вида: <тип результата> <имя функции> (<список формальных параметров>).

Пример.

int g(int n) // — прототип

int f(int n)

{

if (n==1)

return 1;

else return f(n-1) + g(n-1);

}

int g (int n)

{

if (n==1)

return 1;

else

return f(n-2)+2\*g(n-1);

}

**Передача параметров по значению**

Обычно параметры передаются в функцию по значению. Это означает, что числовое значение, которое используется при вызове функции, передается в функцию и там присваивается к некоторой переменной. То есть, по сути функция работает не с самим параметром, а с его копией.

Пример.

int cube (int x)

{

return x\*x\*x;

}

int main()

{

int n=3;

cout<<cube(n);

return 0;

}

При данном вызове создается новая переменная типа int с именем x и в область памяти, выделенную под эту переменную, записывается значение 3. Функция cube работает с новой переменной x.

Параметры, которые используются для приема передаваемых в функцию значений, называют формальными.

Параметры, которые используют для передачи значения в функцию, называют фактическими.

**Передача параметров по ссылке**

Второй способ передачи параметра в функцию — это передача по ссылке. Такой способ позволяет изменять значения некоторого параметра в теле функции.

Для передачи по ссылке используется знак &.

Пример.

void func(int p1, int &p2) // p1 — по значению, p2 — по ссылке

{

p1++;

p2++;

cout<< «Внутри функции»<<p1<< « »<<p2<<endl;

}

int main ()

{

int x=4, y=8;

cout<< «До вызова функции»<<x<< « »<<y<<endl;

func(x,y);

cout<< «После вызова функции»<<x<< « »<<y<<endl;

return 0;

}

Результат работы программы следующий:

* до вызова функции: 4, 8;
* внутри функции: 5, 9;
* после вызова: 4, 9.

При передаче по значению параметр не изменится внутри функции, а при передаче по ссылке — изменится.

В качестве параметров по ссылке можно использовать ссылку или указатель.

Чаще всего указатели используют при передаче массива в качестве параметра функции.

Существуют три способа передачи массива в функцию:

* void func (int a[10]);
* void func (int a[]);
* void func (int \*a).

Однако все 3 рассмотренных способа фактически эквивалентны, поскольку в каждом из них в функцию передается адрес первого элемента массива. Таким образом, все способы эквивалентны и сводятся к последнему.

Пример.

int sum (int \*a, int n)

{

int s=0;

for (int i=0; i<n; i++)

if (a[i]%3==0)

s+=a[i];

return s;

}

int main ()

{

int n;

cout<< «Введи размер массива» << endl;

cin>>n;

int \*a = new int[n];

for (int i=0; i<n; i++)

cin>>a[i];

cout<< sum(a,n)<< endl;

return 0;

}

При передаче параметров по ссылке работа внутри функции ведется не с копией, а с оригиналом параметра.

Параметры, которые передаются по ссылке, позволяют функции возвращать при вызове сразу несколько значений.

Пример.

void square (int &a, int &b)

{

a\*=a;

b\*=b;

}

int main()

{

int x=3, y=5;

square (x,y);

cout<< «x= »<<x<< «y= »<<y<<endl; // в x и y результат работы функции

Параметры по ссылке позволяют эффективно работать с большими объемами данных, поскольку избавляют от необходимости создания копий этих данных.

**Параметры по умолчанию**

В С++ есть возможность устанавливать значения параметров в функции по умолчанию. Эти значения будут использоваться в тех случаях, если соответствующий параметр не будет установлен при вызове функции. Если же параметр установлен при вызове функции, то используется значение из вызова.

Пример.

int func (int x, int y=2)

{

return x+y;

}

int main()

{

cout<<func(4,8) <<endl; //12

cout<< func(3) <<endl; // 5

}

**Рекурсия**

Рекурсивной называется функция, которая в своем теле вызывает саму себя.

Для корректного рекурсивного определения необходимо наличие как минимум одной нерекурсивной ветки. А в рекурсивной части вызов должен осуществляться с измененным параметром: n! = {1, если n=1; (n-1)! \*n, если n>1; fib(n) = {1, если n=1, n=2; fib(n-1) + fib(n-2), при n>2.

Пример.

long fact (int n)

{

if (n==1)

return 1;

else return n\* fact(n-1);

}

int main()

{

cout<< fact(3)<<endl;

return 0;

}

При вызове функции fact(3) формируется активация функции, которую также называют модифицированным телом функции, следующего вида:

int n=3;

if (n==1)

return 1; // 1-ая активация функции

else

return n\*fact(n-1);

Очевидно, что в данном случае срабатывает альтернатива else и результатом работы данной активации является значение 3\*fact(2), которая в данный момент не может быть вычислена.

Работа с данной активацией приостанавливается, и формируется активация функции номер 2, которая выглядит следующим образом:

int n=2;

if (n==1)

return 1; //2-ая активация

else

return n\*fact(n-1);

Результатом работы является значение 2\*fact(1).

Это значение также не может быть вычислено, поэтому работа с данной активацией также приостанавливается и формируется третья активация, которая выглядит следующим образом:

int n=1;

if (n==1)

return 1; // 3-ья активация — дно рекурсии

else

return n\*fact(n-1);

Результат работы этой активации равен 1.

Вычисленное значение 1 подставляется во вторую активацию функции, после чего третья активация уничтожается.

На следующем этапе вычисляется значение из второй активации функции, которая подставляется в первую активацию, после чего вторая активация уничтожается.

Далее вычисляется значение из первой активации функции, которое подставляется в точку вызова функции main, после чего первая активация уничтожается.

В общем случае рекурсивная функция работает дольше чем обычная и ей требуется больше памяти.

Однако существуют такие задачи, которые невозможно решить в общем виде без использования рекурсии.