

Функции в языке Си

Функция — это самостоятельная единица программы, которая спроектирована для реализации конкретной подзадачи.

Функция является подпрограммой, которая может содержаться в основной программе, а может быть создана отдельно (в библиотеке). Каждая функция выполняет в программе определенные действия.

Сигнатура функции определяет правила использования функции. Обычно сигнатура представляет собой описание функции, включающее имя функции, перечень формальных параметров с их типами и тип возвращаемого значения.

Семантика функции определяет способ реализации функции. Обычно представляет собой тело функции.

Определение функции

Каждая функция в языке Си должна быть определена, то есть должны быть указаны:

- ❑ тип возвращаемого значения;
- ❑ имя функции;
- ❑ информация о формальных аргументах;
- ❑ тело функции.

Определение функции имеет следующий синтаксис:

```
ТипВозвращаемогоЗначения ИмяФункции(СписокФормальныхАргументов)
{
    ТелоФункции;
    ...
    return(ВозвращаемоеЗначение);
}
```

Пример: Функция сложения двух вещественных чисел

```
1 float function(float x, float z)
2 {
```

```
3   float y;  
4   y=x+z;  
5   return(y);  
6   }
```

В указанном примере возвращаемое значение имеет тип `float`. В качестве возвращаемого значения в вызывающую функцию передается значение переменной `y`. Формальными аргументами являются значения переменных `x` и `z`.

Если функция не возвращает значения, то тип возвращаемого значения для нее указывается как `void`. При этом операция `return` может быть опущена. Если функция не принимает аргументов, в круглых скобках также указывается `void`.

Различают **системные** (в составе систем программирования) и **собственные** функции.

Системные функции хранятся в стандартных библиотеках, и пользователю не нужно вдаваться в подробности их реализации. Достаточно знать лишь их сигнатуру. Примером системных функций, используемых ранее, являются функции `printf()` и `scanf()`.

Собственные функции — это функции, написанные пользователем для решения конкретной подзадачи.

Разбиение программ на функции дает следующие преимущества:

- ❑ Функцию можно вызвать из различных мест программы, что позволяет избежать повторения программного кода.
- ❑ Одну и ту же функцию можно использовать в разных программах.
- ❑ Функции повышают уровень модульности программы и облегчают ее проектирование.
- ❑ Использование функций облегчает чтение и понимание программы и ускоряет поиск и исправление ошибок.

С точки зрения вызывающей программы функцию можно представить как некий «черный ящик», у которого есть несколько входов и один выход. С точки зрения вызывающей программы неважно, каким образом производится обработка информации внутри функции. Для корректного использования функции достаточно знать лишь ее сигнатуру.

Вызов функции

Общий вид вызова функции

```
Переменная = ИмяФункции(СписокФактическихАргументов);
```

Фактический аргумент — это величина, которая присваивается формальному аргументу при вызове функции. Таким образом, **формальный аргумент** — это переменная в вызываемой функции, а **фактический аргумент** — это конкретное значение, присвоенное этой переменной вызывающей функцией. Фактический аргумент может быть константой, переменной или выражением. Если фактический аргумент представлен в виде выражения, то его значение сначала вычисляется, а затем передается в вызываемую функцию. Если в функцию требуется передать несколько значений, то они записываются через запятую. При этом формальные параметры заменяются значениями фактических параметров в порядке их следования в сигнатуре функции.

Возврат в вызывающую функцию

По окончании выполнения вызываемой функции осуществляется возврат значения в точку ее вызова. Это значение присваивается переменной, тип которой должен соответствовать типу возвращаемого значения функции. Функция может передать в вызывающую программу только одно значение. Для передачи возвращаемого значения в вызывающую функцию используется оператор **return** в одной из форм:

```
return(ВозвращаемоеЗначение);
```

```
return ВозвращаемоеЗначение;
```

Действие оператора следующее: значение выражения, заключенного в скобки, вычисляется и передается в вызывающую функцию. Возвращаемое значение может использоваться в вызывающей программе как часть некоторого выражения.

Оператор **return** также завершает выполнение функции и передает управление следующему оператору в вызывающей функции. Оператор

`return` не обязательно должен находиться в конце тела функции.

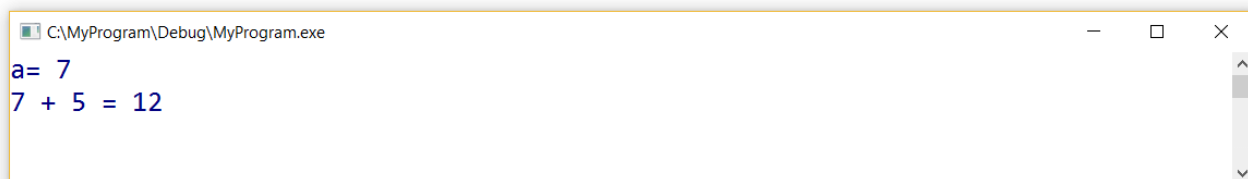
Функции могут и не возвращать значения, а просто выполнять некоторые вычисления. В этом случае указывается пустой тип возвращаемого значения `void`, а оператор `return` может либо отсутствовать, либо не возвращать никакого значения:

```
return;
```

Пример: Посчитать сумму двух чисел.

```
1  #define _CRT_SECURE_NO_WARNINGS // для возможности использования scanf
2  #include <stdio.h>
3  // Функция вычисления суммы двух чисел
4  int sum(int x, int y) // в функцию передаются два целых числа
5  {
6      int k = x + y; // вычисляем сумму чисел и сохраняем в k
7      return k;      // возвращаем значение k
8  }
9  int main()
10 {
11     int a, r;      // описание двух целых переменных
12     printf("a= ");
13     scanf("%d", &a); // вводим a
14     r = sum(a, 5);  // вызов функции: x=a, y=5
15     printf("%d + 5 = %d", a, r); // вывод: a + 5 = r
16     getchar(); getchar(); // мы использовали scanf(),
17     return 0; // поэтому getchar() вызываем дважды
18 }
```

Результат выполнения



```
C:\MyProgram\Debug\MyProgram.exe
a= 7
7 + 5 = 12
```

В языке Си нельзя определять одну функцию внутри другой.

В языке Си нет требования, чтобы семантика функции обязательно предшествовало её вызову. Функции могут определяться как до вызывающей функции, так и после нее. Однако если семантика вызываемой

функции описывается ниже ее вызова, необходимо до вызова функции определить прототип этой функции, содержащий:

- ❑ тип возвращаемого значения;
- ❑ имя функции;
- ❑ типы формальных аргументов в порядке их следования.

Прототип необходим для того, чтобы компилятор мог осуществить проверку соответствия типов передаваемых фактических аргументов типам формальных аргументов. Имена формальных аргументов в прототипе функции могут отсутствовать.

Если в примере выше тело функции сложения чисел разместить после тела функции main, то код будет выглядеть следующим образом:

```
1  #define _CRT_SECURE_NO_WARNINGS // для возможности использования sc
2  #include <stdio.h>
3  int sum(int, int); // сигнатура
4  int main()
5  {
6      int a, r;
7      printf("a= ");
8      scanf("%d", &a);
9      r = sum(a, 5); // вызов функции: x=a, y=5
10     printf("%d + 5 = %d", a, r);
11     getchar(); getchar();
12     return 0;
13 }
14 int sum(int x, int y) // семантика
15 {
16     int k;
17     k = x + y;
18     return(k);
19 }
```

Рекурсивные функции

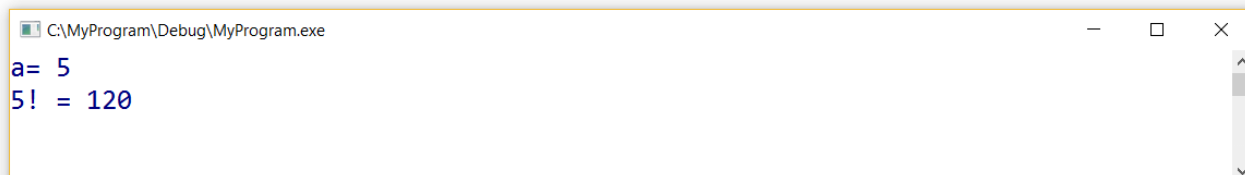
Функция, которая вызывает сама себя, называется **рекурсивной функцией**.

Рекурсия — вызов функции из самой функции.

Пример рекурсивной функции — функция вычисления факториала.

```
1  #define _CRT_SECURE_NO_WARNINGS // для возможности использования sc
2  #include <stdio.h>
3  int fact(int num) // вычисление факториала числа num
4  {
5      if (num <= 1) return 1; // если число не больше 1, возвращаем
6      else return num*fact(num - 1); // рекурсивный вызов для числа на
7  }
8  // Главная функция
9  int main()
10 {
11     int a, r;
12     printf("a= ");
13     scanf("%d", &a);
14     r = fact(a); // вызов функции: num=a
15     printf("%d! = %d", a, r);
16     getchar(); getchar();
17     return 0;
18 }
```

Результат выполнения



```
C:\MyProgram\Debug\MyProgram.exe
a= 5
5! = 120
```

Более подробно рекурсивные функции рассмотрены в [этой статье](#).

Математические функции

Математические функции хранятся в стандартной библиотеке `math.h`. Аргументы большинства математических функций имеют тип `double`. Возвращаемое значение также имеет тип `double`.

Углы в тригонометрических функциях задаются в радианах.

Основные математические функции стандартной библиотеки.

Функция	Описание
<code>int abs(int x)</code>	Модуль целого числа <code>x</code>
<code>double acos(double x)</code>	Арккосинус <code>x</code>

Функция	Описание
<code>double asin(double x)</code>	Арксинус x
<code>double atan(double x)</code>	Арктангенс x
<code>double cos(double x)</code>	Косинус x
<code>double cosh(double x)</code>	Косинус гиперболический x
<code>double exp(double x)</code>	Экспонента x
<code>double fabs(double x)</code>	Модуль вещественного числа
<code>double fmod(double x, double y)</code>	Остаток от деления x/y
<code>double log(double x)</code>	Натуральный логарифм x
<code>double log10(double x)</code>	Десятичный логарифм x
<code>double pow(double x, double y)</code>	x в степени y
<code>double sin(double x)</code>	Синус x
<code>double sinh(double x)</code>	Синус гиперболический x
<code>double sqrt(double x)</code>	Квадратный корень x
<code>double tan(double x)</code>	Тангенс x
<code>double tanh(double x)</code>	Тангенс гиперболический x

Особенности использования функций в языке C++ рассмотрены в [этой статье](#).