Функции

Существенная часть умения программирования - это умение проектировать программный продукт, в частности, разбивать задачу на подзадачи. В языке С нет деления подпрограмм на процедуры и функции, здесь вся программа состоит только из функций.

Функция - это совокупность объявлений и операторов, предназначенная для решения определенной задачи. Каждая функция имеет идентификатор -- имя.

Даже «тело программы» - это функция с именем **main**, главная функция. В программе на С главная функция только одна, т. к. именно с нее, в каком бы месте она не находилась, начинается выполнение программы.

Имя (идентификатор) используется для вызова функции.

При вызове функции ей при помощи аргументов (*формальных параметров*) могут быть переданы некоторые значения (*фактические параметры*), используемые во время выполнения функции.

Функция может возвращать некоторое (одно!) значение. Это возвращаемое значение и есть результат выполнения функции, который при выполнении программы подставляется в точку вызова функции, где бы этот вызов ни встретился.

Допускается создание функций, не имеющих аргументов, а также функций, не возвращающих никаких значений. Действие таких функций может состоять в изменении значений глобальных переменных, выводе на печать текста или других побочных действиях.

Описание функции задает:

1. тип возвращаемого значения;
2. имя функции;
3. типы и число формальных параметров;
4. тело функции.

В определении функции также может быть указан класс памяти.

Пример

```
int digit(unsigned char c)
{
   if (c >= '0' && c <= '9')
       return 1;
   else
```

```
        return 0;
}
```

В данном примере определена функция с именем **digit**, имеющая один параметр с именем **c** и типом **unsigned char**. Функция возвращает целое значение, равное 1, если параметр функции является цифрой, или 0 в противном случае.

В языке С определение функции не обязательно предшествует ее вызову. Определения используемых функций могут находиться как перед местом вызова, так и ниже, или вообще находиться в другом файле.

Однако, чтобы компилятор мог осуществить проверку соответствия типов передаваемых фактических параметров типам формальных параметров, до вызова функции нужно поместить объявление функции --- прототип.

Объявление функции похоже на определение функции, но *тело функции отсутствует*, а *имена формальных параметров могут быть пропущены*. Для функции, определенной в последнем примере, прототип может иметь вид

```
int digit(unsigned char);
```

Имя, тип возвращаемого значения и типы формальных параметров, задаваемые в определении функции, должны соответствовать типу в объявлении этой функции!

Для использования библиотечных функций, т. е. функции предварительно разработанных и записанных в библиотеки, требуется включить в программу их описания (заголовки), что осуществляется с помощью директивы **#include**.

Итак, определение функции имеет следующую форму:

<имя типа> имя функции (<список формальных параметров>) { тело функции }

Имя типа задает тип возвращаемого значения.

Функция не может возвращать массив или функцию!

Если выполнение функции заканчивается оператором **return**, содержащим некоторое выражение, то функция возвращает значение этого выражения. Оно вычисляется, преобразуется, если необходимо, к типу возвращаемого значения и возвращается в точку вызова функции в качестве результата.

Если оператор return не содержит выражения или выполнение функции завершается после выполнения ее последнего оператора (без выполнения оператора return), то возвращаемое значение *не определено*! На практике это означает, что программа поведет себя непредсказуемым образом. Если же по

задумке разработчика функция и не должна ничего возвращать, то в качестве типа результата должен быть использован **void**.

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*.

Список формальных параметров может заканчиваться запятой (,) или запятой с многоточием (, ...), это называется **эллипсис**, и означает, что число аргументов функции переменно. Над дополнительными аргументами не проводится контроль типов! Такие функции лучше не использовать на практике.

Если функция не использует параметров, то наличие круглых скобок обязательно, а вместо списка параметров рекомендуется указать слово void.

Порядок и типы формальных параметров должны быть одинаковыми в определении функции и во всех ее объявлениях. Типы фактических параметров при вызове функции должны быть совместимы с типами соответствующих формальных параметров.

Имена параметров используются в теле функции для доступа к переданным значениям.

Тело функции

Все переменные, объявленные в теле функции без указания класса памяти являются **локальными**. При вызове функции локальным переменным отводится память на стеке и производится их инициализация. Управление передается первому оператору тела функции, начинается ее выполнение, которое продолжается до тех пор, пока не встретится оператор return или последний оператор тела функции. Тогда управление возвращается в точку, следующую за точкой вызова, а локальные переменные исчезают, становятся недоступными. При новом вызове функции для локальных переменных память распределяется вновь, и поэтому *старые значения локальных переменных теряются*.

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 их значения также теряются.

Поскольку передача параметров происходит по значению, в теле функции нельзя изменить значения фактических параметров. Однако, если в качестве параметра передать указатель на некоторую переменную, то используя операцию разыменования можно изменить значение этой переменной.

Пример

```
/* Неправильное использование параметров */
void swap(int x, int y)
{
     int tmp = x;
     x = y;
     y = tmp;
}
```

В данной функции значения x и y, являющихся формальными параметрами, меняются местами, но поскольку эти переменные существуют только внутри функции change, значения фактических параметров, используемых при вызове функции, останутся неизменными.

Для того чтобы менялись местами значения фактических аргументов можно сделать так:

Пример

```
/* Правильное использование параметров */
void swap(int *x, int *y)
{
    int tmp = *x;
    *x = *y;
    *y = tmp;
}
```

При вызове такой функции в качестве фактических параметров должны быть использованы не значения переменных, а их адреса:

```
swap(&a, &b);
```