Тема 1.1 Функции в С

Понятие функции

Опыт показывает, что для написания больших программ лучше пользоваться функциями. В таком случае программа будет состоять из отдельных фрагментов кода. Такой отдельный фрагмент кода и есть функций. Отдельный, потому, что работа отдельной функции не зависит от работы какой-нибудь другой. То есть алгоритм в каждой функции функционально достаточен и не зависим от других алгоритмов программы. Однажды написав функцию, её можно будет с лёгкостью переносить в другие программы. Функция — это фрагмент кода или алгоритм, реализованный на каком-то языке программирования, с целью выполнения определённой последовательности операций. Функции позволяют сделать программу модульной, то есть разделить программу на несколько маленьких подпрограмм (функций), которые в совокупности выполняют поставленную задачу. Еще одно преимущество функций в том, что их можно многократно использовать. Данная возможность позволяет многократно использовать один раз написанный код, что в свою очередь, намного сокращает объем кода программы. 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 являются функции `printf()` и `scanf()`.

– **Собственные** функции - это функции, написанные пользователем для решения конкретной подзадачи.

С использованием функций в языке С связаны три понятия - объявление функции (задание формы обращения к функции), определение функции (описание действий, выполняемых функцией), и вызов функции.

Объявление функции

Под объявлением функции в С понимают объявление прототипа функции. Прототип - это явное объявление функции, которое предшествует определению функции. Тип возвращаемого значения при объявлении функции должен соответствовать типу возвращаемого значения в определении функции. Прототип функции не содержит тела функции, но указывает имя функции, типы аргументов и возвращаемый тип данных. В то время как определение функции описывает, что именно делает функции. Прототип функции воспринимается компилятором как описание её интерфейса.

Если прототип функции не задан, а встретился вызов функции, то строится неявный прототип из анализа формы вызова функции. Тип возвращаемого значения создаваемого прототипа `int`, а список типов и числа параметров функции формируется на основании типов и числа фактических параметров используемых при данном вызове.

Таким образом, прототип функции необходимо задавать в следующих случаях:

1. Функция возвращает значение типа, отличного от `int`.

2. Требуется проинициализировать некоторый указатель на функцию до того, как эта функция будет определена.

Существует несколько способов объявления(определения) функций

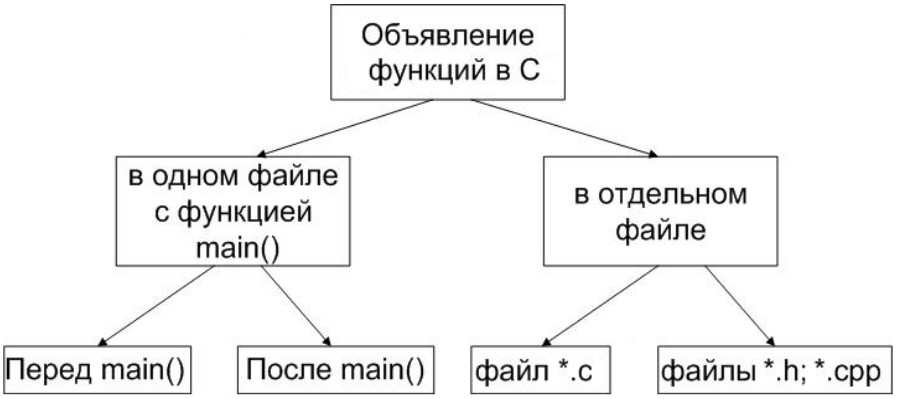

Рисунок 1 - Способы объявления функций

Объявление функций: прототипы *функций*

```
returnDataType functionName( dataType argName1, dataType argName2,
..., dataType argNameN);
```

где,

- `returnDataType` — возвращаемый тип данных
- `functionName` — имя функции
- `dataType` — тип данных
- `argName1..N` — имена параметров функции (количество параметров неограниченно)

Пример объявления функции:

```
/ / Объявление прототипа функции с двумя целыми параметрами
/ / функция принимает два аргумента и возвращает их сумму
int sum(int num1, int num2);
```

В языке С, функции должны быть объявлены до момента их вызова. Вы можете объявить функцию, при этом функция может возвращать значение или — нет, имя функции присваивает программист, типы данных параметров указываются в соответствии с передаваемыми в функцию значениями. Имена аргументов, при объявления прототипов являются необязательными:

```
int sum(int , int ); // тот же прототип функции
```

Определение функции

Определение функции задает тип возвращаемого значения, имя функции, типы и число формальных параметров, а также объявления переменных и операторы, называемые телом функции, и определяющие действие функции.

Синтаксис определения функции:

```
returnDataType functionName( dataType argName1, dataType argName2,
..., dataType argNameN)
{
    // тело функции
}
```

Необязательный спецификатор класса памяти задает класс памяти функции, который может быть `static` или `extern`. Спецификатор типа функции задает тип возвращаемого значения и может задавать любой тип. Если спецификатор типа не задан, то предполагается, что функция возвращает значение типа `int`.

Функция не может возвращать массив или функцию, но может возвращать указатель на любой тип, в том числе и на массив и на функцию. Тип возвращаемого значения, задаваемый в определении функции, должен соответствовать типу в объявлении этой функции.

Функция возвращает значение если ее выполнение заканчивается оператором `return`, содержащим некоторое выражение. Указанное выражение 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 последнего ее оператора (без выполнения оператора `return`), то возвращаемое значение не определено. Для функций, не использующих возвращаемое значение, должен быть использован тип `void`, указывающий на отсутствие возвращаемого значения. Если функция определена как функция, возвращающая некоторое значение, а в операторе `return` при выходе из нее отсутствует выражение, то поведение вызывающей функции после передачи ей управления может быть непредсказуемым.

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 означает, что число аргументов функции переменно. Однако предполагается, что функция имеет, по крайней мере, столько обязательных аргументов, сколько формальных параметров задано перед последней запятой в списке параметров. Такой функции может быть передано большее число аргументов, но над дополнительными аргументами не проводится контроль типов.

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. Тип формального параметра может быть любым основным типом, структурой, объединением, перечислением, указателем или массивом. Если тип формального параметра не указан, то этому параметру присваивается тип `int`.

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 значения этих переменных теряются. Поскольку передача параметров происходит по значению, в теле функции нельзя изменить значения переменных в вызывающей функции, являющихся фактическими параметрами. Однако, если в качестве параметра передать указатель на некоторую переменную, то используя операцию разадресации можно изменить значение этой переменной.

Тело функции - это составной оператор, содержащий операторы, определяющие действие функции. Все переменные, объявленные в теле функции, являются локальными. При вызове функции локальным переменным отводится память в стеке и производится их инициализация. Управление передается первому оператору тела функции и начинается выполнение функции, которое продолжается до тех пор, пока не встретится оператор `return` или последний оператор тела функции. Управление при этом возвращается в точку, следующую за точкой вызова, а локальные переменные становятся недоступными. При новом вызове функции для локальных переменных память распределяется вновь, и поэтому старые значения локальных переменных теряются.

Пример определения функции:

```
// определение функции, которая суммирует два целых числа и воз-
вращает их сумму
int sum(int num1, int num2)
{
    return (num1 + num2);
}
```

В языке С нет требования, чтобы определение функции обязательно предшествовало ее вызову. Определения используемых функций могут следовать за определением функции `main`, перед ним, или находится в другом файле (рисунок 1). Однако для того, чтобы компилятор мог осуществить проверку соответствия типов передаваемых фактических параметров типам формальных параметров до вызова функции нужно поместить объявление (прототип) функции.

```
int sum(int, int); // объявление функции суммирования

int sum(int num1, int num2) // определение функции суммирования
{
    return (num1 + num2);
}
```

Вызов функции

После того, как функция была объявлена и определена, её можно использовать, для этого её нужно вызвать. Вызов функции выполняется следующим образом:

```
funcName( arg1, arg2, ...  );
```

где,

- `funcName` — имя функции;
- `arg1..2`— аргументы функции (фактические параметры)

Поскольку синтаксически имя функции является адресом начала тела функции, в качестве обращения к функции может быть использовано адресное-выражение (в том числе и имя функции или разадресация указателя на функцию), имеющее значение адреса функции.

Список аргументов функции представляет собой список фактических параметров, передаваемых в функцию. Этот список может быть и пустым, но наличие круглых скобок обязательно. Фактический параметр может быть величиной любого основного типа, структурой, объединением, перечислением , выражением или указателем на объект любого типа. Массив и функция не могут быть использованы в качестве фактических параметров, но можно использовать указатели на эти объекты. 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.

Пример программы с использованием функции `multiplication`:

```
#include <stdio.h>

int multiplication( int num1, int num2 ); //объявление (прототип)
функции
int main()
{
  int num1;
  int num2;

  printf( "Введите два  числа для умножения: " );
  scanf( "%d", &num1 );
  scanf( "%d", &num2 );
  printf( "Результат умножения %d\n", multiplication( num1, num2 )
); // вызов функции
  getchar();
  return 0;
}
int multiplication(int num1, int num2) // определение функции
{
  return num1 * num2;
}
```

Библиотеки функций и их подключение

В программах на языке С широко используются, так называемые, библиотечные функции, т.е. функции предварительно разработанные и записанные в библиотеки. Прототипы библиотечных функций находятся в специальных заголовочных файлах, поставляемых вместе с библиотеками в составе систем программирования, и включаются в программу с помощью директивы `#include`. Например, чтобы воспользоваться функцией, которая возводит некоторое число в степень, нужно подключить заголовочный файл `<cmath>` и запустить функцию `pow()` в теле программы.

```
// подключение заголовочного файла <cmath> который содержит прото-
типы основных математических функций
#include <cmath>

int main(int argc, char* argv[])
{
    float power = pow(3.14,2); // запуск функции возведения числа
в степень
    return 0;
}
```