A function is a group of statements that together perform a task. Every C program has at least one function, which is main(), and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.

A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

The C standard library provides numerous built-in functions that your program can call. For example, strcat() to concatenate two strings, memcpy() to copy one memory location to another location, and many more functions.

A function can also be referred as a method or a sub-routine or a procedure, etc.

The general form of a function definition in C programming language is as follows:

```
return_type function_name( parameter list )
{
   body of the function
}
```

A function definition in C programming consists of a *function header* and a *function body*. Here are all the parts of a function:

- Return Type: A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.
- Function Name: This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- Parameters: A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- Function Body: The function body contains a collection of statements that define what the function does.

Example

Given below is the source code for a function called max(). This function takes two parameters num1 and num2 and returns the maximum value between the two:

```
/* function returning the max between two numbers */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;

}
```

A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts:

```
return_type function_name( parameter list );
```

For the above defined function max(), the function declaration is as follows:

```
int max(int num1, int num2);
```

Parameter names are not important in function declaration only their type is required, so the following is also a valid declaration:

```
int max(int, int);
```

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.

When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value. For example:

```
#include <stdio.h>

/* function declaration */
int max(int num1, int num2);

int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;

   /* calling a function to get max value */
   ret = max(a, b);

   printf( "Max value is : %d\n", ret );

   return 0;
}

/* function returning the max between two numbers */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

We have kept max() along with main() and compiled the source code. While running the final executable, it would produce the following result:

```
Max value is : 200
```

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the formal parameters of the function.

Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

While calling a function, there are two ways in which arguments can be passed to a function:

Call Type & Description
Call by value method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
Call by reference method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

By default, C uses call by value to pass arguments. In general, it means the code within a function cannot alter the arguments used to call the function.

A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed. There are three places where variables can be declared in C programming language:

- Inside a function or a block which is called local variables.
- Outside of all functions which is called global variables.
- In the definition of function parameters which are called formal parameters.

Understand local and global variables, and formal parameters.

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. The following example shows how local variables are used. Here all the variables a, b, and C are local to main() function.

```
#include <stdio.h>

int main () {

  /* local variable declaration */
  int a, b;
  int c;

  /* actual initialization */
  a = 10;
  b = 20;
  c = a + b;

  printf("value of a = %d, b = %d and c = %d\n", a, b, c);

  return 0;
}
```

Global variables are defined outside a function, usually on top of the program. Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. The following program show how global variables are used in a program.

```
#include <stdio.h>

/* global variable declaration */
int g;

int main () {

  /* local variable declaration */
  int a, b;

  /* actual initialization */
  a = 10;
  b = 20;
  g = a + b;

  printf("value of a = %d, b = %d and g = %d\n", a, b, g);

  return 0;
}
```

A program can have same name for local and global variables but the value of local variable inside a function will take preference. Here is an example:

```
#include <stdio.h>

/* global variable declaration */
int g = 20;

int main () {

  /* local variable declaration */
  int g = 10;

  printf("value of g = %d\n",  g);

  return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
value of g = 10
```

Formal parameters, are treated as local variables with-in a function and they take precedence over global variables. Following is an example:

```
#include <stdio.h>

/* global variable declaration */
int a = 20;

int main () {

  /* local variable declaration in main function */
  int a = 10;
  int b = 20;
  int c = 0;

  printf("value of a in main() = %d\n",  a);
  c = sum( a, b);
```

```
  printf ("value of c in main() = %d\n",  c);

  return 0;
}

/* function to add two integers */
int sum(int a, int b) {

   printf("value of a in sum() = %d\n",  a);
   printf("value of b in sum() = %d\n",  b);

   return a + b;
}
```

When the above code is compiled and executed, it produces the following result:

```
value of a in main() = 10
value of a in sum() = 10
value of b in sum() = 20
value of c in main() = 30
```

When a local variable is defined, it is not initialized by the system, you must initialize it yourself. Global variables are initialized automatically by the system when you define them as follows:

Data Type	Initial Default Value
int	0
char	'\0'
float	0
double	0
pointer	NULL

It is a good programming practice to initialize variables properly, otherwise your program may produce unexpected results, because uninitialized variables will take some garbage value already available at their memory location.