CHAPTER 5

SUBPROGRAMS IN C

A sub-program is a series of C statements that perform a specific task in a program. A subprogram can be called within another procedure. Every C program has at least one function, which is **main().** A C program can be divided up into separate functions.

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, function strcat() to concatenate two strings, function memcpy() to copy one memory location to another location and many more functions.

A function is known with various names like a method or a sub-routine or a procedure, etc. However, a **function** returns a value while a **procedure** doesn't: it just executes commands.

A Subprogram is:

- a part of a program that performs one or more related tasks
- has its own name
- written as an independent part of the program

Benefits of using sub-procedures in programming are:

- Sub-programs help to break programs into several but logical sections. The smaller programs/routines make **programming**, **debugging** and subsequent **maintenance** easier.
- They also help in **coding repeated operations** such as frequently used calculations, text etc thus making programming less repetitive and faster.
- Procedures used in one program can act as building blocks for other programs with slight modifications i.e. **code re-use**.
- Programmers working on large projects can divide the workload by making different functions.

TYPES OF FUNCTIONS

Basically, there are two types of functions in C on basis of whether it is defined by user or not.

- Library function
- User defined function

LIBRARY FUNCTION

Library functions are the in-built function in C programming system. For example:

printf()

- **`printf()`** is used for displaying output in C.

scanf()

- **`scanf()`** is used for taking input in C.

USER DEFINED FUNCTION

C allows programmers to define their own function according to their requirements known as user defined functions.

How user-defined function works in C Programming?

```
#include <stdio.h>
void function_name(){
................
................
}
int main(){
...........
...........
function_name();
...........
...........
}
```

As mentioned earlier, every C program begins from `main()` and program starts executing the codes inside `main()` function. When the control of program reaches to `function_name()` inside `main()` function. The control of program jumps to `void function_name()` and executes the codes inside it. When, all the codes inside that user-defined function are executed, control of the program jumps to the statement just after `function_name()` from where it is called. Analyze the figure below for understanding the concept of function in C programming.

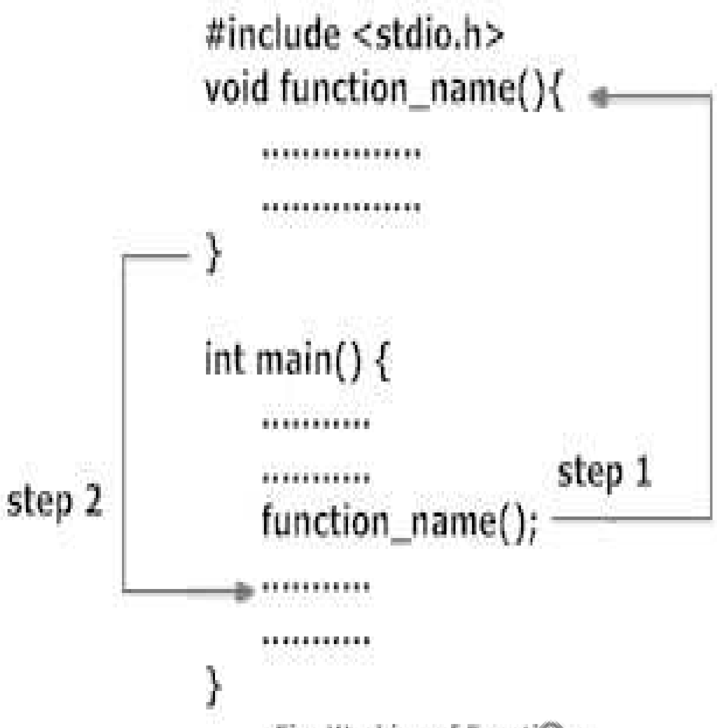


Fig: Working of Functions

Remember, the function name is an identifier and should be unique.

DEFINING A FUNCTION

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 language consists of a function header and a function body. Here are all the parts of a function:

1. **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**.
2. **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the **function signature**.
3. **Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the **formal 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.
4. **Function Body:** The function body contains a collection of statements that define what the function does.

Example

Following is the source code for a function called max(). This function takes two parameters num1 and num2 and returns the maximum 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;
}
```

FUNCTION DECLARATIONS

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(), following is the function declaration:

int max(int num1, int num2);

Parameter names are not important in function declaration; only their type is required, so the following is also 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.

CALLING A 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, program control is transferred to the called function. A called function performs defined task, and when its return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program. Therefore, the calling program is suspended during execution of the called subprogram.
To call a function, you simply need to pass the required parameters along with function name, and if function returns a value, then you can store 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;
}
```

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

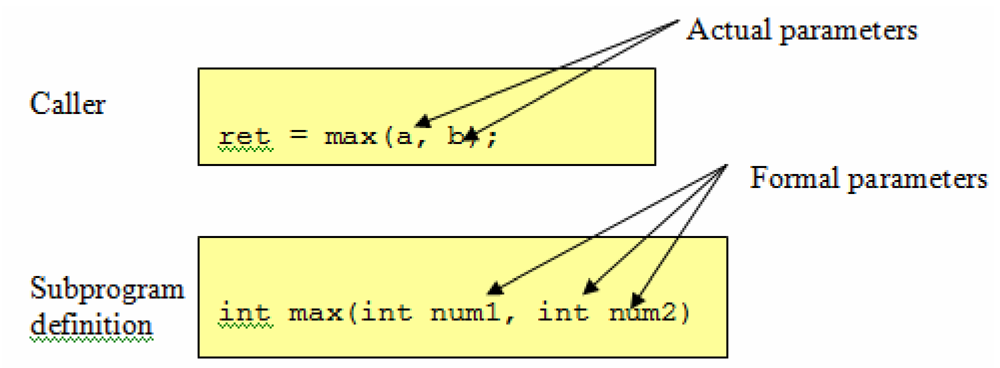
While calling a function, there are two ways that arguments can be passed to a function:

Call Type	**Description**
Call by value	This 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	This 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, this means that code within a function cannot alter the arguments used to call the function and above mentioned example while calling max() function used the same method.

FUNCTION ARGUMENTS

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.
The formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.
A formal parameter is a dummy variable listed in the subprogram header and used in the subprogram. An actual parameter represents a value used in the subprogram call statement.



When max() is called, we pass it the arguments which the function uses as the values of ret. This process is called **parameter passing**.
*****

Parameters refers to the list of variables in a method declaration. *Arguments* are the actual values that are passed in when the method is invoked. When you invoke a method, the arguments used must match the declaration's parameters in type and order.

TYPES OF VARIABLES

The Programming language C has two main variable types

- Local Variables
- Global Variables

LOCAL VARIABLES

A local variable is a variable that is declared inside a function.

- Local variables scope is confined within the block or function where it is defined. Local variables must always be defined at the top of a block.
- When execution of the block starts the variable is available, and when the block ends the variable 'dies'.

GLOBAL VARIABLES

Global variable is defined at the top of the program file and it can be visible and modified by any function that may reference it. Global variables are declared outside **all** functions.