Tutorial Link https://codequotient.com/tutorials/Functions in C/59fdb047e63d6b7fd5debf5b

TUTORIAL

Functions in C

Chapter

1. Functions in C

Function is a piece of code that performs a specific task. Even though it is possible to write each and every line of code in the main function, but writing different functions for different tasks is easy way to programming. Functions provide many benefits while writing programs like reusability, easy understanding, easy debugging etc. Till now we generally worked with the main function and other standard library functions like printf(), scanf() etc. but C also allows to write our own functions. So function calling can be depicted as below code snippet: -

```
#include <stdio.h>
void functionName()
{
    ... .. ...
    ... .. ...
}               // Control will be back to the main function
(which is known as calling function).

int main()
{
    ... .. ...
    functionName();     // Control will transfer to
functionName definition
    ... .. ...
    ... .. ...
}
```

Functions need the three things in your programs: -

- Function declaration: To indicate the compiler that there exists a function may be defined later which is used at any instruction during execution.
- Function definition: It is the definition of the function, where the working of the function is written.
- Function calling: It is the place where a function is called for serving the purpose it is intended to serve.

Functions can be declared by following syntax: -

```
return_type    function_name( parameter_list );
```

where return_type is any valid C data type (int, char, float etc.) which resembles the returning value type from the function. If a function does not return anything then it should be equal to void. function_name is the name of the function which is required every time you want to refer it. It must bind with the rules of a valid identifier in C, so you have to take care while choosing the names for your functions. parameter_list is a list of variables (if any) called parameters, which the function needs to start work with. This list must define the datatype of each variable. For example,

```
int power_function(int number, int power);
```

is a power_function which accepts two int arguments and will return a integer value when finish its execution. If we write everything in main function then sooner or later main function will become heavy and difficult to manage or debug. So to provide the reusability and better understanding of the problem it is better to design/code in modules/functions so later, it will become easy for us to solve the whole problem. Also, functions work like encapsulation of some functionality in it. So that later when we need to solve a task for which a function is already written then we can focus on the use of function only, not on how the function is solving the problem. So we can focus on main problem later. Following is an example program which demonstrates the use of functions: -

```c
int addNumbers(int a,int b)
{
  // function definition
  int result;
  result = a+b;
  return result;                // return value to callee function
}
```

Output of the above program will be as below, if called with a=20, b=20: -

```
sum = 40
```

The arguments the functions receive are called the formal arguments/parameters which works like any other local variable for the function. These variables will be stored on the stack allocated to memory in program allocation. When a function starts its execution it

will be given space on the stack area of memory (more on program memory is explain in this article). In the above code the actual arguments passed to the function are n1 and n2, which are accepted by the function in formal parameters a and b. So when the function is called a and b will be created in the function activation record and as soon as the function ends, they will be removed from the memory.