Functions in C

A function is a set of statements that take inputs, do some specific computation and produces output.

The idea is to put some commonly or repeatedly done task together and make a function so that instead of writing the same code again and again for different inputs, we can call the function.

Example:
Below is a simple C/C++ program to demonstrate functions.

```
#include <stdio.h>

// An example function that takes
two parameters 'x' and 'y'
// as input and returns max of
two input numbers
int max(int x, int y)
{
    if (x > y)
      return x;
    else
      return y;
}

// main function that doesn't
receive any parameter and
// returns integer.
int main(void)
{
    int a = 10, b = 20;
```

```
    // Calling above function to
find max of 'a' and 'b'
    int m = max(a, b);

    printf("m is %d", m);
    return 0;
}
```

Output:

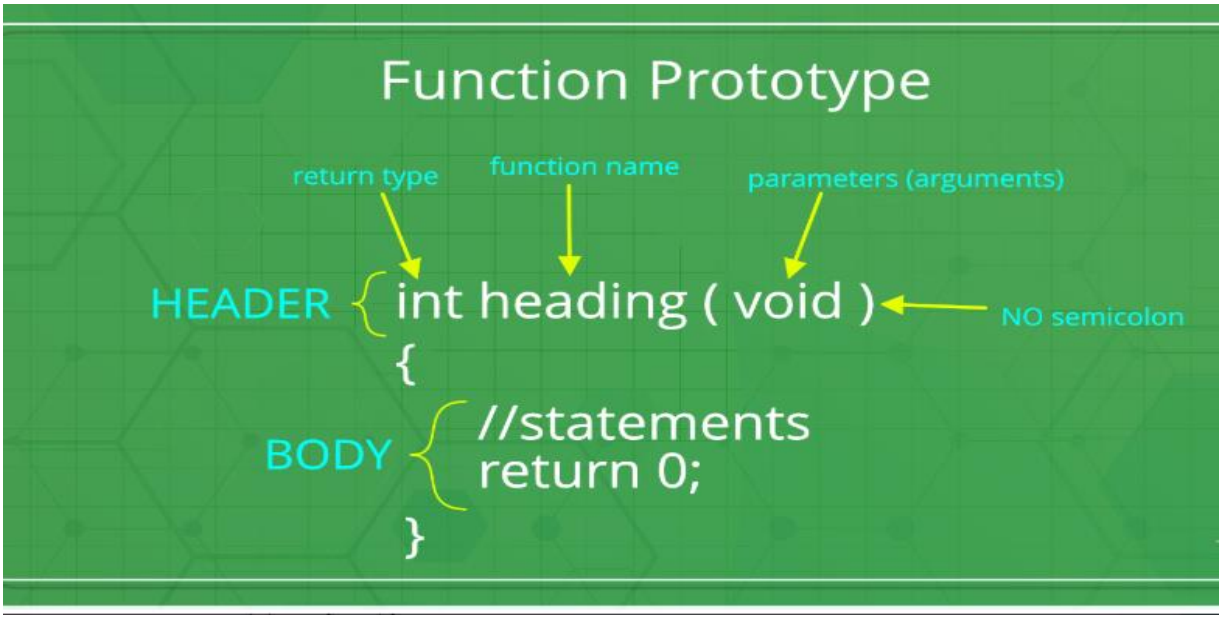
```
m is 20
```

Why do we need functions?

- Functions help us in reducing code redundancy. If functionality is performed at multiple places in software, then rather than writing the same code, again and again, we create a function and call it everywhere. This also helps in maintenance as we have to change at one place if we make future changes to the functionality.
- Functions make code modular. Consider a big file having many lines of codes. It becomes really simple to read and use the code if the code is divided into functions.
- Functions provide abstraction. For example, we can use library functions without worrying about their internal working.

Function Declaration

A function declaration tells the compiler about the number of parameters function takes, data-types of parameters and return type of function. Putting parameter names in function declaration is optional in the function declaration, but it is necessary to put them in the definition. Below are an example of function declarations. (parameter names are not there in below declarations)



```
// A function that takes two
integers as parameters
// and returns an integer
int max(int, int);

// A function that takes a int
pointer and an int variable as
parameters
// and returns an pointer of type
int
int *swap(int*,int);

// A function that takes a charas
parameters
// and returns an reference
variable
char *call(char b);

// A function that takes a char and
an int as parameters
// and returns an integer
int fun(char, int);
```

It is always recommended to declare a function before it is used
(See this, this and this for details)
In C, we can do both declaration and definition at the same place, like done in the above example program.

C also allows to declare and define functions separately, this is especially needed in case of library functions. The library functions are declared in header files and defined in library files. Below is an example declaration.

Parameter Passing to functions
The parameters passed to function are called ***actual parameters***. For example, in the above program 10 and 20 are actual parameters.
The parameters received by function are called ***formal parameters***. For example, in the

above program x and y are formal parameters.
There are two most popular ways to pass parameters.
Pass by Value: In this parameter passing method, values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of caller.
Pass by Reference Both actual and formal parameters refer to same locations, so any changes made inside the function are actually reflected in actual parameters of caller.

Parameters are always passed by value in C. For example. in the below code, value of x is not modified using the function fun().

```
#include <stdio.h>
void fun(int x)
{
    x = 30;
}

int main(void)
{
    int x = 20;
    fun(x);
    printf("x = %d", x);
    return 0;
}
```

Output:

```
x = 20
```

However, in C, we can use pointers to get the effect of pass by reference. For example, consider the below program. The function fun() expects a pointer ptr to an integer (or an address of an integer). It modifies the value at the address ptr. The dereference operator * is used to access the value at an address. In the statement '*ptr = 30', value at address ptr is changed to 30. The address operator & is used to get the address of a variable of any data type. In the function call statement 'fun(&x)', the address of x is passed so that x can be modified using its address.

```
# include <stdio.h>
void fun(int *ptr)
```

```
{
    *ptr = 30;
}

int main()
{
  int x = 20;
  fun(&x);
  printf("x = %d", x);

  return 0;
}
```

Output:

```
x = 30
```

Following are some important points about functions in C.

1) Every C program has a function called main() that is called by operating system when a user runs the program.

2) Every function has a return type. If a function doesn't return any value, then void is used as return type. Moreover, if the return type of the function is void, we still can use return statement in the body of function definition by not specifying any constant, variable, etc. with it, by only mentioning the 'return;' statement which would symbolise the termination of the function as shown below:

```
void function name(int a)
{
.......  //Function Body
return;  //Function execution would
get terminated
}
```

3) In C, functions can return any type except arrays and functions. We can get around this limitation by returning pointer to array or pointer to function.

4) Empty parameter list in C mean that the parameter list is not specified and function can be called with any parameters. In C, it is not a good idea to declare a function like fun(). To declare a function that can only be called without any parameter, we should use "void fun(void)".

As a side note, in C++, empty list means function can only be called without any parameter. In C++, both void fun() and void fun(void) are same.

5)If in a C program, a function is called before its declaration then the C compiler automatically assumes the declaration of that function in the following way:

int function name();

And in that case if the return type of that function is different than INT ,compiler would show an error.