User Define Function

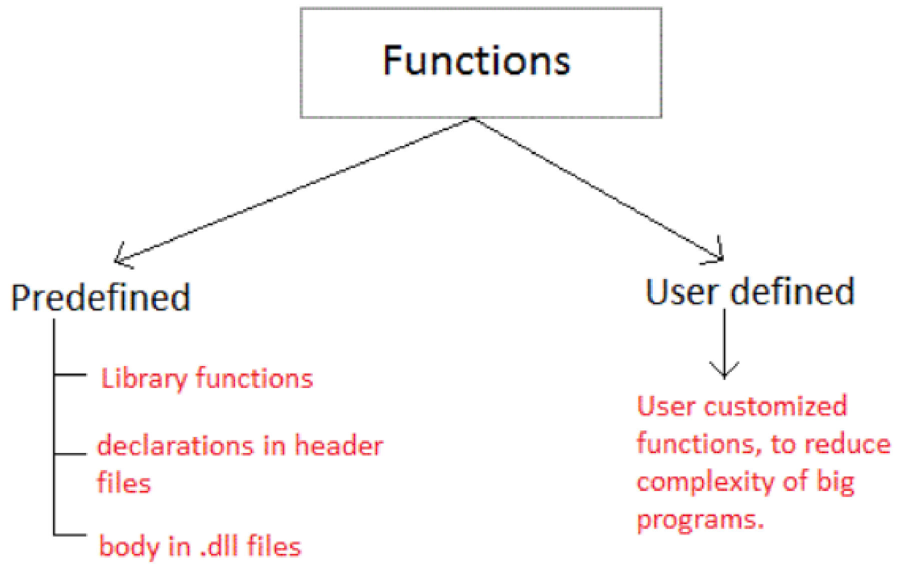
A **function** is a block of code that performs a particular task.

There are many situations where we might need to write same line of code for more than once in a program. This may lead to unnecessary repetition of code, bugs and even becomes boring for the programmer. So, C language provides an approach in which you can declare and define a group of statements once in the form of a function and it can be called and used whenever required.

These functions defined by the user are also known as **User-defined Functions**

C functions can be classified into two categories,

1. **Library functions**
2. **User-defined functions**


Functions
Predefined
User defined
Library functions
declarations in header files
body in .dll files
User customized functions, to reduce complexity of big programs.

Library functions are those functions which are already defined in C library, example **printf()**, **scanf()**, **strcat()** etc. You just need to include appropriate header files to use these functions. These are already declared and defined in C libraries.

A **User-defined functions** on the other hand, are those functions which are defined by the user at the time of writing program. These functions are made for code reusability and for saving time and space.

Benefits of Using Functions

1. It provides modularity to your program's structure.
2. It makes your code reusable. You just have to call the function by its name to use it, wherever required.
3. In case of large programs with thousands of code lines, debugging and editing becomes easier if you use functions.
4. It makes the program more readable and easier to understand.

Function Declaration

General syntax for function declaration is,

returntype functionName(type1 parameter1, type2 parameter2,...);

Like any variable or an array, a function must also be declared before its used. Function declaration informs the compiler about the function name, parameters is accept, and its return type. The actual body of the function can be defined separately. It's also called as **Function Prototyping**. Function declaration consists of 4 parts.

- return type
- function name
- parameter list
- terminating semicolon

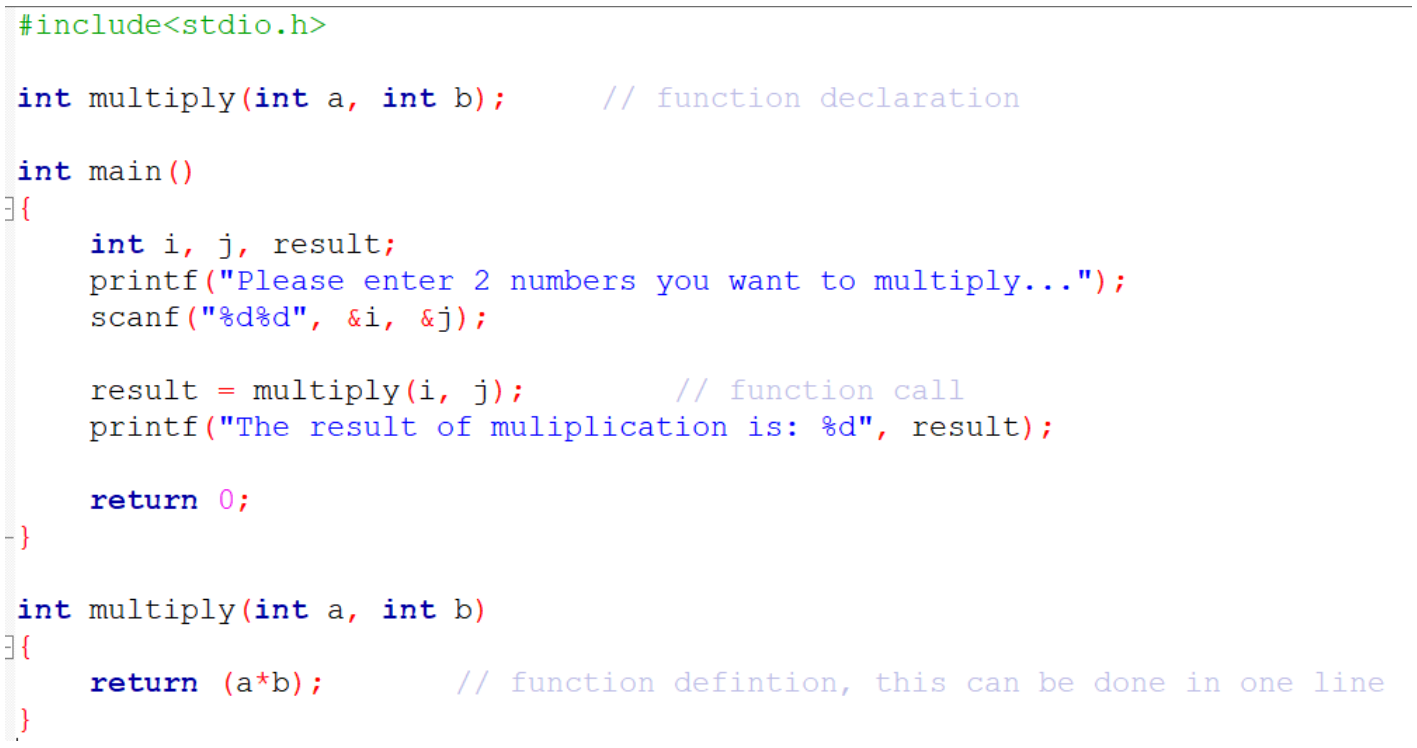
Return Type − A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.

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 1:



```c
#include<stdio.h>

int multiply(int a, int b);      // function declaration

int main()
{
    int i, j, result;
    printf("Please enter 2 numbers you want to multiply...");
    scanf("%d%d", &i, &j);

    result = multiply(i, j);        // function call
    printf("The result of muliplication is: %d", result);

    return 0;
}

int multiply(int a, int b)
{
    return (a*b);       // function defintion, this can be done in one line
}
```

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(), 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.

Calling a Function

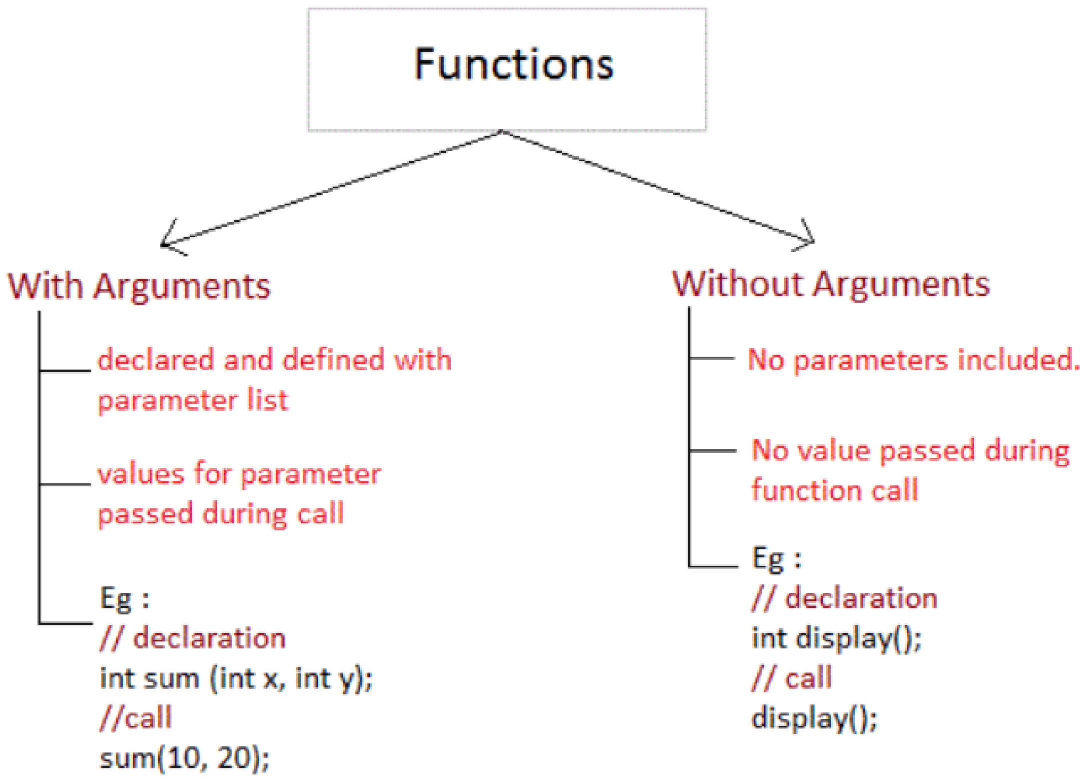
While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.

When a program calls a function, 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.

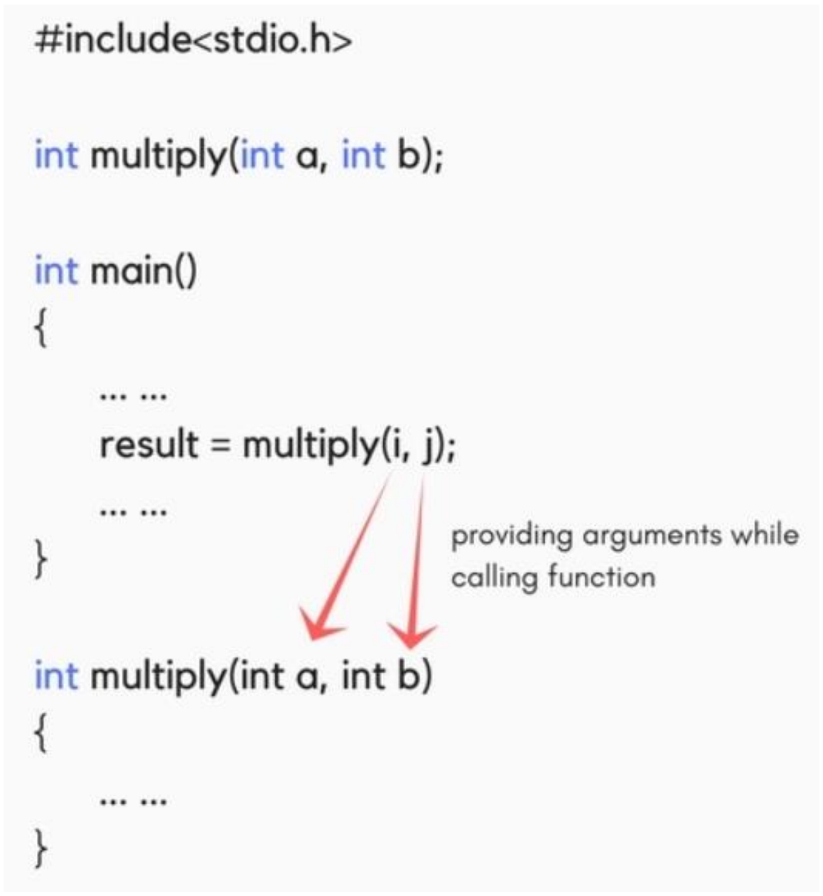
To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value.

Passing Arguments to a function

Arguments are the values specified during the function call, for which the formal parameters are declared while defining the function.



It is possible to have a function with parameters but no return type. It is not necessary, that if a function accepts parameter(s), it must return a result too.

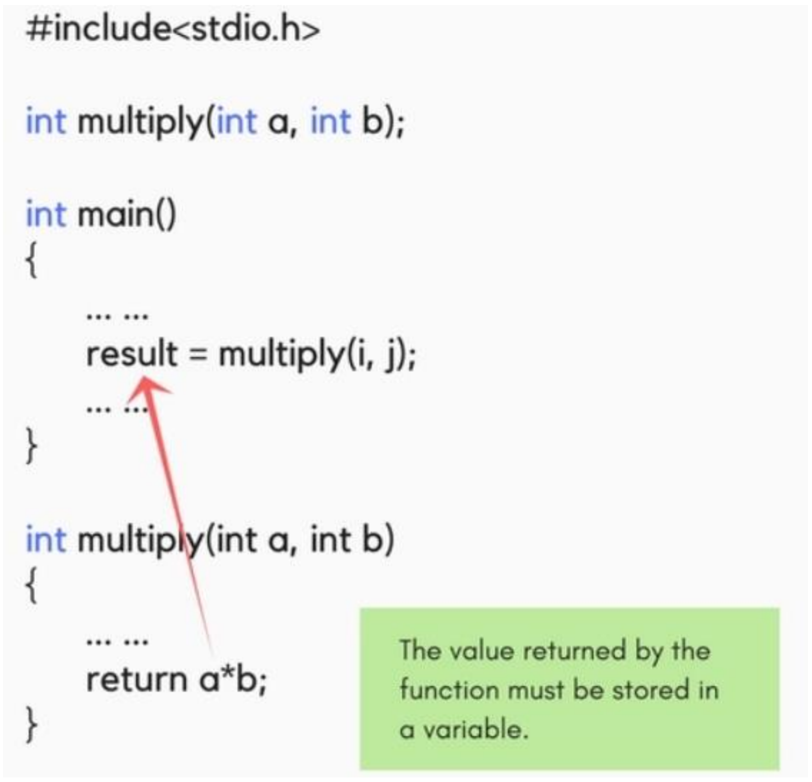


```
#include<stdio.h>

int multiply(int a, int b);

int main()
{
    ... ...
    result = multiply(i, j);
    ... ...
}

int multiply(int a, int b)
{
    ... ...
}
```

While declaring the function, we have declared two parameters a and b of type int. Therefore, while calling that function, we need to pass two arguments, else we will get compilation error. And the two arguments passed should be received in the function definition, which means that the function header in the function definition should have the two parameters to hold the argument values. These received arguments are also known as **formal parameters**. The name of the variables while declaring, calling and defining a function can be different.

Returning a value from function

A function may or may not return a result. But if it does, we must use the return statement to output the result. return statement also ends the function execution, hence it must be the last statement of any function. If you write any statement after the return statement, it won't be executed.



```
#include<stdio.h>

int multiply(int a, int b);

int main()
{
    ... ...
    result = multiply(i, j);
    ... ...
}

int multiply(int a, int b)
{
    ... ...
    return a*b;
}
```

The value returned by the function must be stored in a variable.

The datatype of the value returned using the return statement should be same as the return type mentioned at function declaration and definition. If any of it mismatches, you will get compilation error.