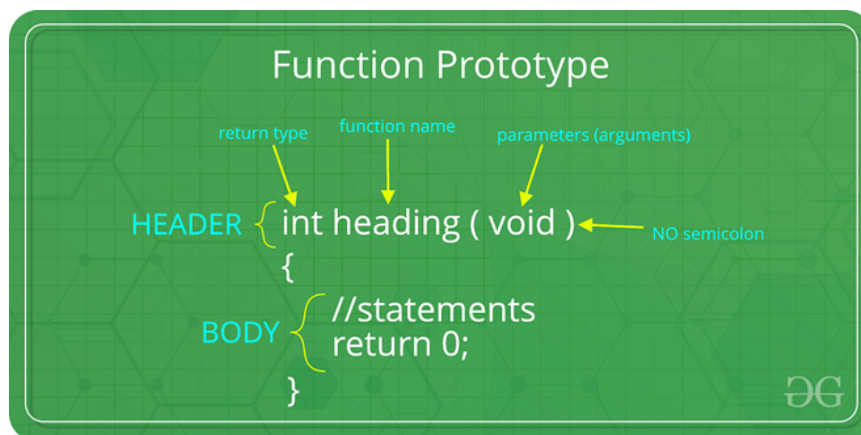


Function declaration & definition

Function Declaration

A function declaration tells the compiler about the number of parameters function takes data-types of parameters, and returns the type of function. Putting parameter names in the 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 the below declarations)



Example:

```
// C++ Program to show function that takes
// two integers as parameters and returns
// an integer
int max(int, int);

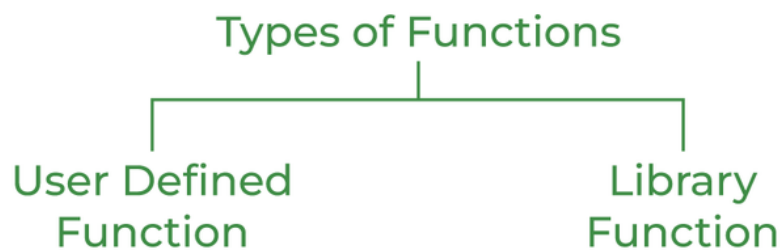
// A function that takes an int
// pointer and an int variable
// as parameters and returns
// a pointer of type int
int* swap(int*, int);

// A function that takes
// a char as parameter and
// returns a 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);
```

Types of Functions



User Defined Function

User Defined functions are user/customer-defined blocks of code specially customized to reduce the complexity of big programs. They are also commonly known as “***tailor-made functions***” which are built only to satisfy the condition in which the user is facing issues meanwhile reducing the complexity of the whole program.

Library Function

Library functions are also called “***builtin Functions***“. These functions are a part of a compiler package that is already defined and consists of a special function with special and different meanings. Builtin Function gives us an edge as we can directly use them without defining them whereas in the user-defined function we have to declare and define a function before using them.

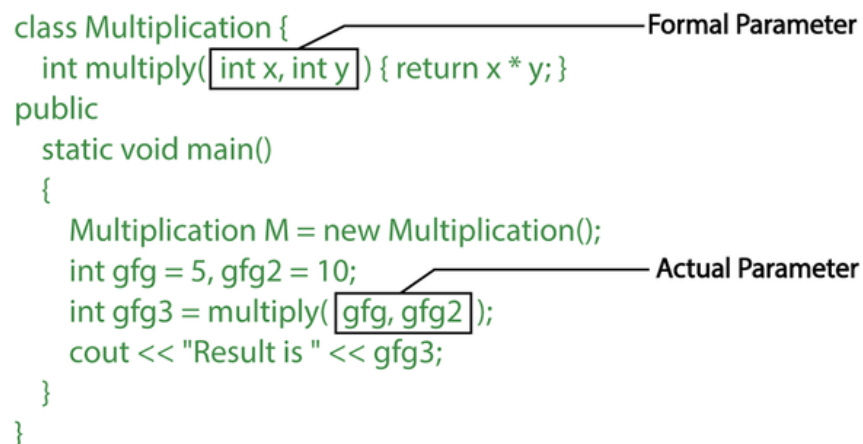
For Example: `sqrt()`, `setw()`, `strcat()`, etc.

Parameter Passing to Functions

The parameters passed to function are called ***actual parameters***. For example, in the program below, 5 and 10 are actual parameters.

The parameters received by the function are called **formal parameters**. For example, in the above program x and y are formal parameters.

```
class Multiplication {  
    int multiply(int x, int y) { return x * y; }  
public  
    static void main()  
    {  
        Multiplication M = new Multiplication();  
        int gfg = 5, gfg2 = 10;  
        int gfg3 = multiply(gfg, gfg2);  
        cout << "Result is " << gfg3;  
    }  
}
```



There are two most popular ways to pass parameters:

1. **Pass by Value:** In this parameter passing method, values of actual parameters are copied to the 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 the actual parameters of the caller.
2. **Pass by Reference:** Both actual and formal parameters refer to the same locations, so any changes made inside the function are actually reflected in the actual parameters of the caller.

Function Definition

Pass by reference is used where the value of x is not modified using the function fun().

```
// C++ Program to demonstrate function definition  
#include <iostream>  
using namespace std;  
  
void fun(int x)  
{  
    // definition of  
    // function  
    x = 30;  
}
```

```
int main()
{
    int x = 20;
    fun(x);
    cout << "x = " << x;
    return 0;
}
```

Output

```
x = 20
```