Introduction to functions

Overview

A Function is a collection of statements designed to perform a specific task. The function is like a black box that can take specific input(s) as its parameters and output a value that is the return value. A function is created to use it as many times as needed just by using the function's name. You don't need to type the function's statements whenever required; you just have to call the function. A function is a block of code that only runs when called.

Every C++ program has at least one function, which is main().

Defining a function: It provides information about the function attributes such as its name, return type, parameters, and body.

Syntax to define a function:

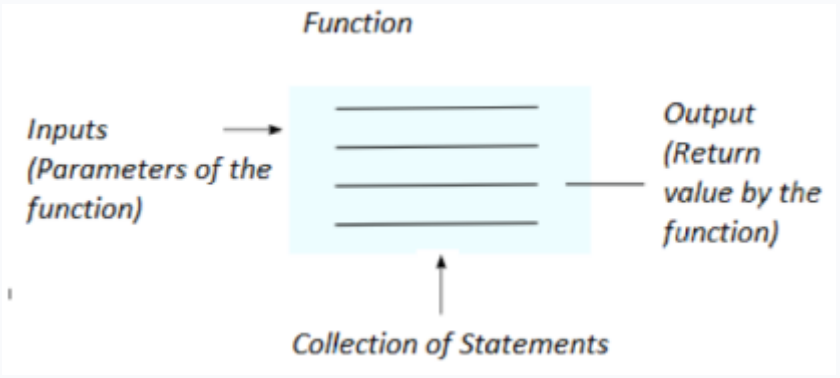
```
return_type functionName(parameter1, parameter2, ...) {
    // function body
}
```

Here,

- 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, So in that case, the return_type is the keyword void. Some other return_type can be int, string, vector, float, etc.
- 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 an actual parameter or

argument. The parameter list refers to the type, order, and the, number of function parameters. 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: In this example, we define a function that accepts two parameters and returns the maximum of two numbers.

```
int max(int x, int y) {

    if (x > y)
        return x;
    else
        return y;
}

Here, the return type is int, so it means it will return an integer
value where the function will be called.
```

C++ Function Prototype

A function prototype is a declaration of the function that tells the program about the type of the value returned by the function and the number and type of arguments.

Function prototyping is one very useful feature of the C++ function. A function prototype describes the functional interface to the compiler by giving details such as the number and type of arguments and return values.

Function Declaration: When a function is defined before the main() function in the program, then function declaration is not required, but writing the function after the main() function requires function declaration first, else there will be a compilation error.

Syntax to declare a function:

```
return_type function_name( parameter list );
```

Example:

```
int sum(int num1,int num2);

Can be declared as

int sum(int,int)
```

Example:

```
#include <iostream>
using namespace std;

//Function declaration
int sum(int, int);

int main() {
    //Calling the function
    cout << sum(50, 60);
    return 0;
}
/* Function is defined after main*/

int sum(int num1, int num2) {
    int num3 = num1 + num2;
    return num3;
}
```

Output:

```
110
```

Function Calling in C++: Once we declare a function, we use it to perform some specific task. When we call a function, the program controls transfer to the called function.

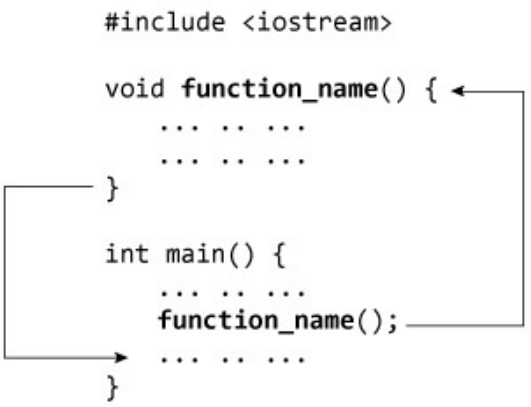
Syntax to call a function:

```
function_name( parameters );
```

Example:

```
#include<iostream>
using namespace std;

int max(int x, int y) {
    if (x > y)
        return x;
    else
        return y;
}

int main() {
    int a = 10;
    int b = 20;

    // Method Calling
    int maximum = max(a, b);
    cout << maximum;
    return 0;
}

Output:
20
```

How does function calling work?



Example:

```
#include <iostream>
using namespace std;

int findSum(int a, int b) {
    int sum = a + b;
    return sum;
}

int main() {
    cout << findSum(50, 60);
    return 0;
}

Output:
110
```

The function being called is called callee(here it is findsum function), and the function which calls the callee is called the caller (here, the main function is the caller).

When a function is called, program control goes to the function's entry point. The entry point is where the function is defined. So focus now shifts to the callee, and the caller function goes in a paused state.

Why do we need function?

- **Reusability:** Once a function is defined, it can be used repeatedly. You can call the function as many times as needed, which saves work. Consider that you are

required to find out the area of the circle. Now either you can apply the formula every time to get the circle area or make a function to find the circle area and invoke the function whenever needed.

- **Neat code:** A code created with a function is easy to read and dry run. You don't need to repeatedly type the same statements; instead, you can invoke the function whenever needed.
- **Modularisation:** Functions help in modularizing code. Modularisation means divides the code into small modules, each performing a specific task. Functions allow in doing so as they are the program's tiny fragments designed to perform the specified task.
- **Easy Debugging:** It is easy to find and correct the error in function compared to raw code without function where you must correct the error everywhere the specific task of the function is performed.