Functions

Functions are the main part of top-down structured programming. We break the code into small pieces and make functions of that code. Functions could be called multiple or several times to provide reusability and modularity to the C++ program.

Functions are also called procedures or subroutines or methods and they are often defined to perform a specific task. And that makes functions a group of code put together and given a name that can be called anytime without writing the whole code again and again in a program.

Advantages of Functions

- The use of functions allows us to avoid re-writing the same logic or code over and over again.
- With the help of functions, we can divide the work among the programmers.
- We can easily debug or can find bugs in any program using functions.
- They make code readable and less complex.

Aspects of a function

- **Declaration**: This is where a function is declared to tell the compiler about its existence.
- **Definition**: A function is defined to get some task executed. (It means when we define a function, we write the whole code of that function and this is where the actual implementation of the function is done).
- **Call**: This is where a function is called in order to be used.

Function Prototype in C++

The function prototype is the template of the function which tells the details of the function which include its name and parameters to the compiler. Function prototypes help us to define a function after the function call.

Example of a function prototype,

// Function prototypereturn_datatype function_name(datatype_1 a, datatype_2 b);

Types of functions

Library functions:

Library functions are pre-defined functions in C++ Language. These are the functions that are included in C++ header files prior to any other part of the code in order to be used.

E.g. sqrt(), abs(), etc.

User-defined functions

User-defined functions are functions created by the programmer for the reduction of the complexity of a program. Rather, these are functions that the user creates as per the requirements of a program.

E.g. Any function created by the programmer.

Functions Parameters

A function receives information that is passed to them as a parameter. Parameters act as variables inside the function.

Parameters are specified collectively inside the parentheses after the function name. Parameters inside the parentheses are comma-separated.

We have different names for different parameters.

Formal Parameters

So, the variable which is declared in the function is called a formal parameter or simply, a parameter. For example, variables a and b are formal parameters.

```
int sum(int a, int b){   //function body}
```

Actual Parameters

The values which are passed to the function are called actual parameters or simply, arguments. For example, the values num1 and num2 are arguments.

```
int sum(int a, int b);


int main()
{
   int num1 = 5;
   int num2 = 6;
   sum(num1, num2); // actual parameters
}
```

A function doesn't need to have parameters or it should necessarily return some value.

Methods

Now, there are methods using which arguments are sent to the function. They are:

Call by Value in C++

Call by value is a method in C++ to pass the values to the function arguments. In the case of call by value, the copies of actual parameters are sent to the formal parameter, which means that if we change the values inside the function, it will not affect the actual values.

An example that demonstrates the call by value method is:

```
#include <iostream>

using namespace std;


void swap(int a, int b)

{

   int temp = a;

   a = b;

   b = temp;

}


int main()

{

   int x = 5, y = 6;

   cout << "The value of x is " << x << " and the value of y is " << y << endl;

   swap(x, y);

   cout << "The value of x is " << x << " and the value of y is " << y << endl;

}
```

Output:

The value of x is 5 and the value of y is 6The value of x is 5 and the value of y is 6

Call by Pointer in C++

A call by pointer is a method in C++ to pass the values to the function arguments. In the case of call by pointer, the address of actual parameters is sent to the formal parameter, which means that if we change the values inside the function, it will affect the actual values.

An example that demonstrates the call by pointer method is:

```
#include <iostream>

using namespace std;


void swap(int *a, int *b)

{

   int temp = *a;

   *a = *b;

   *b = temp;

}


int main()

{

   int x = 5, y = 6;

   cout << "The value of x is " << x << " and the value of y is " << y << endl;

   swap(&x, &y);

   cout << "The value of x is " << x << " and the value of y is " << y << endl;

}
```

Output:

```
The value of x is 5 and the value of y is 6The value of x is 6 and the value of y is 5
```

Call by Reference in C++

Call by reference is a method in C++ to pass the values to the function arguments. In the case of call by reference, the reference of actual parameters is sent to the formal parameter, which means that if we change the values inside the function, it will affect the actual values.

An example that demonstrates the call by reference method is:

```
#include <iostream>

using namespace std;


void swap(int &a, int &b)

{

  int temp = a;

  a = b;

  b = temp;

}


int main()

{

  int x = 5, y = 6;

  cout << "The value of x is " << x << " and the value of y is " << y << endl;

  swap(x, y);

  cout << "The value of x is " << x << " and the value of y is " << y << endl;

}
```

Output:

The value of x is 5 and the value of y is 6

The value of x is 6 and the value of y is 5

Two special methods that are often used by programmers to have their program work efficiently are:

Default Arguments in C++

Default arguments are those values which are used by the function if we don't input our value as a parameter. It is recommended to write default arguments after the other arguments.

An example using default argument:

```
int sum(int a = 5, int b);
```

Constant Arguments in C++

Constant arguments are used when you don't want your values to be changed or modified by the function. The const keyword is used to make the parameters non-modifiable.

An example using constant argument:

```
int sum(const int a, int b);
```

Recursion

When a function calls itself, it is called recursion and the function which is calling itself is called a recursive function. The recursive function consists of a base condition and a recursive condition.

Recursive functions must be designed with a base case to make sure the recursion stops, otherwise, they are bound to execute forever and that's not what you want. The case in which the function doesn't recur is called the base case. For example, when we try to find the factorial of a number using recursion, the case when our number becomes smaller than 1 is the base case.

An example of a recursive function is the function for calculating the factorial of a number.

```
int factorial(int n){
  if (n == 0 || n == 1){
    return 1;
  }
  return n * factorial(n - 1);
}
```

Function Overloading

Function overloading is a process to make more than one function with the same name but different parameters, numbers, or sequences. Now, there are a few conditions and any number of functions with the same name following any of these are called overloaded.

Same name but different data type of parameters

Example:

float sum(int a, int b);float sum(float a, float b);

Same name but a different number of parameters

Example:

float sum(int a, int b);float sum(int a, int b, int c);

Same name but different parameter sequence

Example:

float sum(int a, float b);float sum(float a, int b);

Exact matches are always preferred while looking for a function that has the same set of parameters.