Lecture 5: FUNCTIONS in C++

A function is a group of statements used to perform a certain operation. It is called from some point in the main program or another function. A function is a block of code that performs a specific task. Every C++ program has at least one function, which is the main () function. There are two types of Functions:

1. Standard Library Functions: Predefined in C++.

2. User-defined Function: Created by users:

The functions that are created by the C++ programmer so that we can use them many times. It reduces the complexity of a big program and optimizes the code.

Create a Function

C++ provides some pre-defined functions, such as main(), which is used to execute code. But you can also create your own functions to perform certain actions. To create a function, specify the name of the function, followed by parentheses **()**:

Syntax

```
void myFunction() {
  // code to be executed
}
```

Example Explained

- myFunction () is the name of the function.
- void means that the function does not have a return value.
- inside the function (the body), add code that defines what the function should do.

Call a Function

Declared functions are not executed immediately. They are "saved for later use", and will be executed later when they are called.

To call a function, write the function's name followed by two parentheses () and a semicolon;

In the following example, myFunction() is used to print a text (the action), when it is called:

Example

Inside main, call myFunction ():

```
// Create a function
void myFunction() {
  cout << "I just got executed!";
}

int main() {
  myFunction(); // call the function
  return 0;
}
// Outputs "I just got executed!"
```

A function can be called multiple times:

Example

```
void myFunction () {
  cout << "I just got executed!\n";
}

int main() {
  myFunction();
  myFunction();
  myFunction();
  return 0;
}

// I just got executed!
// I just got executed!
// I just got executed!
```

➢ Parameters and Arguments

Information can be passed to functions as a parameter. Parameters act as variables inside the function.

Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma:

Syntax

```
void functionName(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

The following example has a function that takes a string called fname as a parameter. When the function is called, we pass along a first name, which is used inside the function to print the full name:

Example

```
void myFunction(string fname) {
  cout << fname << " Brown\n";
}

int main() {
  myFunction("Liam");
  myFunction("Jenny");
  myFunction("Anja");
  return 0;
}

// Liam Brown
// Jenny Brown
// Anja Brown
```

When a **parameter** is passed to the function, it is called an **argument**. So, from the example above: fname is a **parameter**, while Liam, Jenny and Anja are **arguments**.

➢ FUNCTIONS RETURN A VALUE

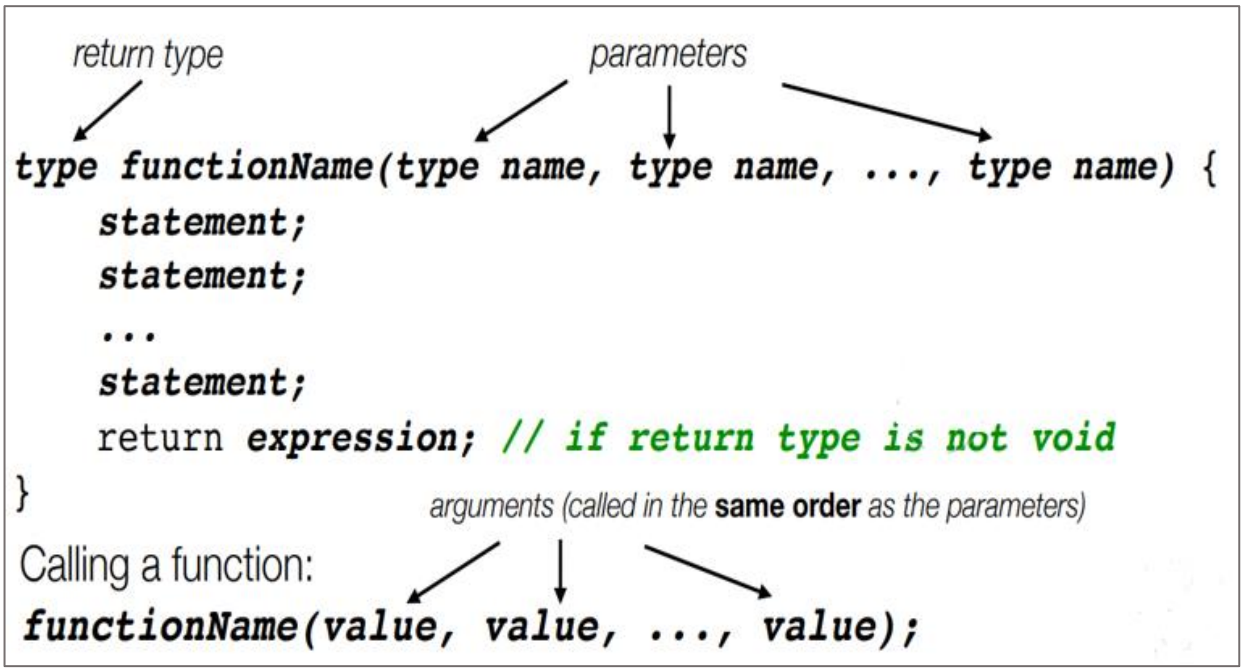
The void keyword, used in the previous examples, indicates that the function should not return a value. If you want the function to return a value, you can use a data type (such as int, string, etc.) instead of void, and use the return keyword inside the function

```
type function_name (type parameter1, type parameter2, ……)
{
statements;
 return value;
}
```

The syntax is illustrated below:

In the above definition, the first word is the type of the function, it is the type of data it returns. The second item is the name of the function. (Type parameter1, type parameter2, ……) are called arguments of the function.

ACCESSING A FUNCTION (The Call of the Function)



```
type functionName(type name, type name, ..., type name) {
    statement;
    statement;
    ...
    statement;
    return expression; // if return type is not void
}
functionName(value, value, ..., value);
```

The main program calls the function by declaring a variable followed by the function name and its arguments, as follows:

type variable_name=function_name(arguments)

Example

```
int myFunction(int x) {
  return 5 + x;
}

int main() {
  cout << myFunction(3);
  return 0;
}

// Outputs 8 (5 + 3)
```

This example returns the sum of a function with **two parameters**:

Example

```
int myFunction(int x, int y) {
  return x + y;
}

int main() {
  cout << myFunction(5, 3);
  return 0;
}

// Outputs 8 (5 + 3)
```

You can also store the result in a variable:

Example

```
int myFunction(int x, int y) {
  return x + y;
}

int main() {
  int z = myFunction(5, 3);
  cout << z;
  return 0;
}
// Outputs 8 (5 + 3)
```

Ex: Write a program includes a function receives a character and returns its type (number, lowercase alphabet, uppercase alphabet, or symbol)

```
#include <iostream>
using namespace std;
int chaletter( char c)
{
 if(c >= 'A' && c <= 'Z')
 return (1);
 else if(c >= 'a' && c <= 'z')
 return (2);
 else if (c>='0' && c<='9')
 return (0);
 else return(3);
}
main()
{
int type;
 char ch;
 cout << "Enter the character:"<<endl;
 cin >> ch;
 type = chaletter(ch );
 switch(type)
 {
 case 0: cout << "It is a number"; break;
 case 1: cout << "It is an uppercase alphabet"; break;
 case 2: cout << "It is a lowercase alphabet"; break;
 default: cout<<" It is a symbol";
 }
 return 0;
}
```

Standard Library Functions

Come along with the compiler and is presented in the C++ header file such as ceil(), floor(), sqrt(), etc.

Functions of <cmath> header file.
The C++ <cmath> header file declares a set of functions to perform mathematical operations.

C++ ceil()

Return ceiling value of the number

The ceil() function in C++ returns the smallest possible integer value which is **greater** than or **equal** to the given argument.

ceil() Parameters

The ceil() function takes a single argument whose ceiling value is computed.

Example: ceil() function for double, float and long double types

```
   #include <iostream>
   #include <cmath>
   using namespace std;
   int main()
   {
        double x = 10.25,
        result;result =
        ceil(x);
        cout << "Ceil of " << x << " = " << result <<
        endl;return 0;
 }
```

Output

```
\\ Ceil of 10.25 = 11
```

C++ floor()

The floor() function in C++ returns the largest possible integer value which is **less** than or **equal** to the given argument.

floor() Parameters

The floor() function takes a single argument whose floor value is computed.

floor() Return value

The floor() function returns the largest possible integer value which is **less than** or **equal** to the given argument.

Example: How floor() works in C++?

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
      double x = 10.25, result;
      result =floor(x);
      x = -34.251;
      result = floor(x);
      cout << "Floor of " << x << " = " << result << endl;
      x = 0.71;
      result = floor(x);
      cout << "Floor of " << x << " = " << result << endl;

      return 0;
}
```

Output

```
\\Floor of -34.251 = -35

\\Floor of 0.71 = 0
```

C++ exp ()
returns exponential (e) raised to a number

The exp () function in C++ returns the exponential.

exp () Parameters

The exp () function takes a single mandatory argument and can be any value i.e.negative, positive, or zero.

exp() Return value

The exp() function returns the value in the range of [0, ∞].

Example: How exp() function works in C++?

```
#include <iostream>
#include <cmath>

using namespace std;
int main()
{
        double x = 2.19, result;

        result = exp(x);
        cout << "exp(x) = " << result << endl;

        return 0;
}
```

Output

```
 exp(x) = 8.93521
```

C++ sqrt()

Computes the Square Root of a Number

The sqrt() function in C++ returns the square root of a number.

sqrt() Parameters

The sqrt() function takes a single non-negative argument.

If the negative argument is passed to the sqrt() function, a domain error occurs.

Example: How sqrt() works in C++?

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
        double x = 10.25, result;
        result = sqrt(x);
        cout << "Square root of " << x << " is "<< result<< endl;
        return 0;
}
```

Output

```
Square root of 10.25 is 3.20156
```

C++ fmax()

returns largest among two arguments passed

The fmax() function in C++ takes two arguments and returns the largest amongthem. If one of the argument is NaN, the other argument is returned.

fmax() Parameters

x: The first argument of fmax().
y: The second argument of fmax().

Example: fmax() function for arguments of the same type

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

int main()
{
    double x = -2.05, y = NAN, result;
    result = fmax(x, y);
    cout << "fmax(x, y) = " << result << endl;
    return 0;
}
```

Output

```
fmax(x, y) = -2.05
```

C++ fmin()

returns smallest among two given arguments

The fmin() function in C++ takes two arguments and returns the smallest among them. If one of the argument is NaN, the other argument is returned.

Example: fmax() function for arguments of the same type

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

int main()
{
    double x =2.05, y = 2, result;
    result = fmin(x, y);
    cout << "fmin(x, y) = " << result << endl;
    return 0;
}
```

Output

```
fmin(x, y) = 2
```