# EXPERIMENT NO 12

## User Defined Functions in C++

**Objectives**:

- To understand the concepts of functions.
- To understand what the function prototype is and how that is different from the function definition.
- Convert the code processing in the main function to a function called from the main function.
- Create functions with multiple parameters.
- The mechanisms for passing information between functions and returning results.
- Understand the difference between pass by value and reference.
- To know about function overloading

**Equipment required**:

- Dev-C++/Eclipse/Visual Studio installed in PC/Windows

## DISCUSSION

## 1. Pre-Lab

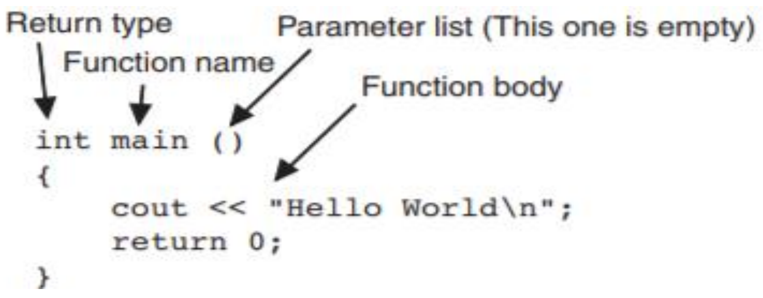
### Functions

A function is a collection of statements that performs a specific task. So far you have experienced functions as you have created a function named “main” in every program you have written. Functions are commonly used to break a problem down into small manageable pieces. This approach is sometimes called divide and conquer because a large problem is divided into several smaller problems that are easily solved.

#### Defining and Calling Functions

A function call is a statement that causes a function to execute. A function definition contains the statements that make up the function. When creating a function, you must write its definition. All function definitions have the following parts:

1. **Name:** You should give each function a descriptive name. In general, the same rules that apply to variable names also apply to function names.
2. **Parameter list:** The program can send data into a function. The parameter list is a list of variables that hold the values being passed to the function.
3. **Body:** The body of a function is the set of statements that perform the function’s operation. They are enclosed in a set of braces.
4. **Return type:** A function can send a value to the part of the program that executed it. The return type is the data type of the value that is sent from the function.



```
int main ()
{
    cout << "Hello World\n";
    return 0;
}
```

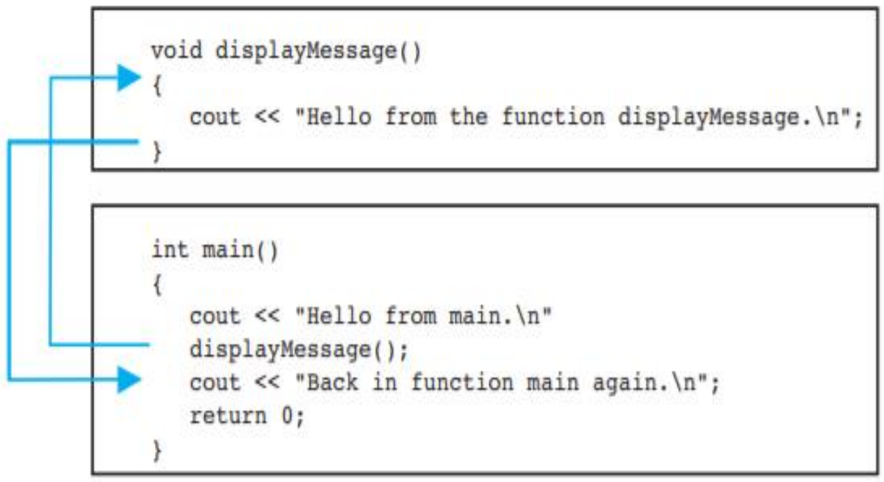
**Void Functions**

It is not necessary for all functions to return a value; however, some functions simply perform one or more statements, which follows terminate. These are called void functions. An example is shown below.

```
void displayMessage()
{
   cout << "Hello from the function displayMessage.\n";
}
```

**Calling a Function**

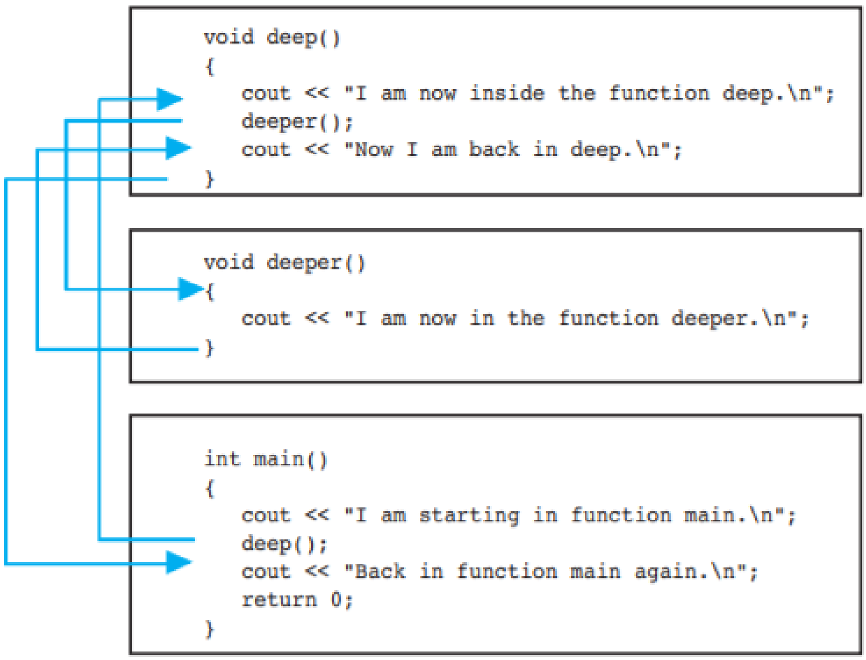
A function is executed when it is called. The "main" function is called automatically when a program starts, but all other functions must be executed by function call statements. When a function is called, the program branches to that function and executes the statements in its body.



```
void displayMessage()
{
   cout << "Hello from the function displayMessage.\n";
}

int main()
{
   cout << "Hello from main.\n"
   displayMessage();
   cout << "Back in function main again.\n";
   return 0;
}
```

In the above example, **void displayMessage()** represent the Function Header (Definition) while the 2nd line in the "main" function i.e. **displayMessage();** represent the function call. When this line executes the program move towards the function definition and execute the whole body, then it comes back to the main function.

Functions may also be called in a hierarchical, or layered, fashion. It means a function can call another function too, example is shown below:



```
void deep()
{
   cout << "I am now inside the function deep.\n";
   deeper();
   cout << "Now I am back in deep.\n";
}

void deeper()
{
   cout << "I am now in the function deeper.\n";
}

int main()
{
   cout << "I am starting in function main.\n";
   deep();
   cout << "Back in function main again.\n";
   return 0;
}
```

**Function prototype**

A function prototype eliminates the need to place a function definition before all calls to the function. You must place either the function definition or the function prototype ahead of all calls to the function. Otherwise, the program will not compile.

**Sending data into a Function**

When a function is called, the program may send values into the function. Values that are sent into a function are called arguments. In the following statement the function pow is being called and two arguments, 2.0 and 4.0, are passed to it:

```
result = pow (2.0, 4.0);
```

By using parameters, you can design your own functions that accept data this way. A parameter is a special variable that holds a value being passed into a function. The values can be passed to the function as “value” or as “reference”.

**Pass by Value**

***Example***

```
void displayValue (int x); //function prototype
int main ()    // main function
{
int x = 2;
displayValue(x);  //function is called and the parameter x is passed to the function definition
return 0;
}

void displayValue (int x) //function definition
{
cout<< "The value is " <<x<<endl; }
```

Here the value “x” passed by the main function is stored in another “x”. Keep in mind this “x” is different from the one in the main function. Any change we make to this “x” will not change the value of “x” present in the “main” function. Suppose we update the “x” value here as, x = 3, then “x” has value 3 in this function only but in the main function “x” will not get updated, it will still stay as x = 2 because both are different. We can give a different name to the “x” in the function definition to avoid confusion. In the same program, we can also write the function definition line as,

```
void displayValue (int num) //function definition
```

Now the variable sent by the main function is “x” whose value is copied in the variable “num”.

**Pass by Reference**

***Example***

```
void displayValue (int& x); //function prototype
int main ()    // main function
{
int x = 2;
displayValue(x);  //function is called and the parameter “x” is passed to the function definition
}

void displayValue (int& x) //function definition
{
cout<< "The value is " <<x<<endl;
}
```

Notice this “&” with the data type in function definition. Here the value “x” passed by main function is stored in another “x” but by writing “&” gives an address to the “x” in the main function. So, if I change this “x” value within the function it will also change the value of “x” in the main function. Even if we had stored it in another variable, it still has the address of the main function variable “x”, so any change we make into this variable will update the variable in the main function also.

```
void displayValue (int& num) //function definition
```

Now the variable sent by the main function is “x” whose value is copied in the variable “num”. If we change the value of “num” here the value of “x” in the main function will also change as “num” has the address of “x”.
This “&” should be written in both function definition as well as prototype.

**The Return Statement and Returning a Value from a Function**

The return statement causes a function to end immediately. A function may send a value back to the part of the program that called the function this is known as returning a value. Here is an example of a function that returns an int value:

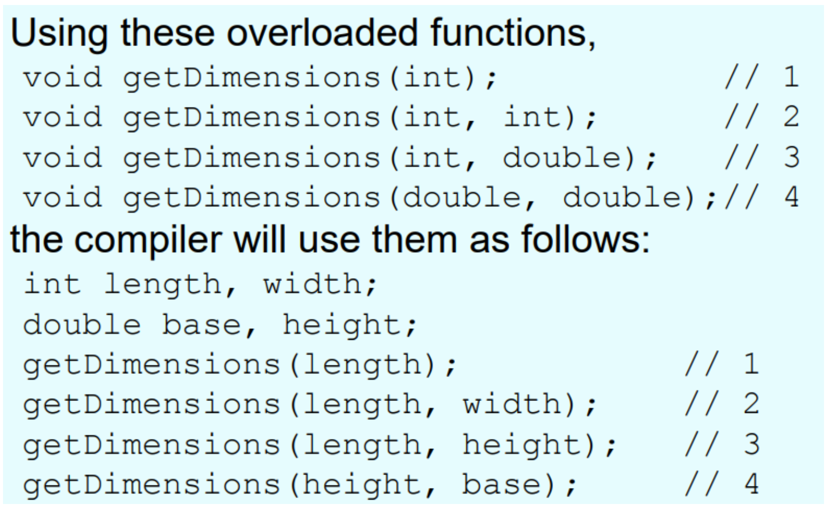
```
int sum(int num1, int num2)
{
   int result;

   result = num1 + num2;
   return result;
}
```

A function can also return a Boolean value instead of integer or double or character. Then we must use “bool” in the function definition instead of “int”

**Function Overloading**

Overloaded functions have the same name but different parameter lists. They can be used to create functions that perform the same task but take different parameter types or different number of parameters. The compiler will determine which version of function to call by argument and parameter lists.

***Example***



Using these overloaded functions,

```
void getDimensions(int);            // 1
void getDimensions(int, int);       // 2
void getDimensions(int, double);    // 3
void getDimensions(double, double);// 4
```

the compiler will use them as follows:

```
int length, width;
double base, height;
getDimensions(length);           // 1
getDimensions(length, width);    // 2
getDimensions(length, height);   // 3
getDimensions(height, base);     // 4
```

In the above example, all the functions being called are different. Although their names are same but the parameters or their data types are change, that makes them a different function.

## 2. Post-Lab (Lab Tasks)

1. Write a main program that has 4 functions. The 4 functions should perform the following operations:

   **Sum:** It should add the 2 numbers and display the addition result
   **Multiply:** It should multiply the 2 numbers and display the multiplication result.
   **Subtract:** It should subtract the 2 numbers and return the subtraction result.
   **Divide:** It should take 2 numbers from user and display the division result.

2. Write a function that asks the user to enter the radius of the circle and then returns that number as a float. Write another function that takes this radius as input and returns the area of circle.

3. Using Function overloading create two functions, one should find the maximum number among 4 numbers and the other should find the maximum number among 3 numbers. Both should return the result to main and then display at the output.

4. Write a function courseGrade. This function takes as a parameter an int value specifying the score for a course and returns the grade, a value of type char, for the course.

5. Write a function that finds the minimum number among 3 default arguments of function. Now find the minimum number again by calling the same function, with 2 default arguments and 1 passed by value. Do it again for all arguments passed by value.

6. Write a function swap that takes two numbers as input parameters and swap them, display the swapped numbers from the main function.

7. Write a program that inputs two integers. It passes first integer to a function that calculates and returns its square. It passes second integer to another function that calculates and returns its cube. The main functions adds both returned values and displays the result.

8. Write a program that inputs two integers in main function and passes the values to a function. The function finds and returns the greatest common divisor. The main function then displays the returned value.

9. Write a main function that call a function named countCall. Every time this function is called you should display a message “I have been called 1 time” etc. If the function is called 3 times the output should be: **(Use pass by Reference)**

   ```
   I have been called 1 time.
   I have been called 2 time.
   I have been called 3 time.
   ```

   **Note:** (If your Roll Number is 201795, it should call the function 5 times etc.)

END