LAB 11: Functions

Objective(s):

To Understand about:

Apply user defined functions in C++

1. Function declaration, calling and definition.
2. Functions with Return Value

CLOs: CLO1, CLO4

INTRODUCTION:

The best way to develop and maintain a large program is to construct it from small, simple pieces, or components. This technique is called divide and conquer. Functions allow you to modularize a program by separating its tasks into self-contained units. Functions you write are referred to as user-defined functions or programmer-defined functions. The statements in function bodies are written only once, are reused from perhaps several locations in a program and are hidden from other functions.

Explanation

A –***Function Prototype*** tells the compiler about a function‘s name, function‘s return type and function parameters.

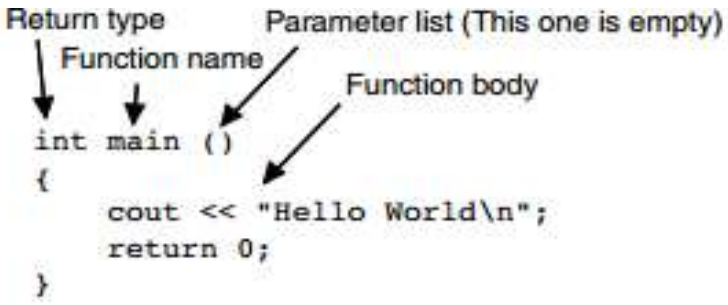
A –***Function Call*** calls a function, program control is transferred to the called function (Function definition). A called function performs defined tasks and when its return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.
To call a function, you simply need to pass the required parameters along with function name, and if function returns a value, then you can store returned value.

A –***Function Definition*** consists of a function header and a function body. Here are all the parts of a function Definition:
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. In this case, the return_type is the keyword void.

Function Name: This is the actual name of the function.

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 actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

Function Body: The function body contains a collection of statements that define what statements to execute.



The line in the definition that reads `int main()` is called the *function header.*

Example:

```
// This program has two functions: main and displayMessage
#include <iostream>
using namespace std;

//*****************************************
// Definition of function displayMessage  *
// This function displays a greeting.     *
//*****************************************

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

//*****************************************
// Function main                          *
//*****************************************

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

Program Output:

```
Program Output
Hello from main.
Hello from the function displayMessage.
Back in function main again.
```

Function Prototypes:
A function prototype eliminates the need to place a function definition before all calls to the function.
Example:

```
// This program has three functions: main, first, and second.
#include <iostream>
using namespace std;

// Function Prototypes
void first();
void second();

int main()
{
   cout << "I am starting in function main.\n";
   first();    // Call function first
   second();   // Call function second
   cout << "Back in function main again.\n";
   return 0;
}

//*************************************
// Definition of function first.      *
// This function displays a message.  *
//*************************************

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

//*************************************
// Definition of function second.     *
// This function displays a message.  *
//*************************************

void second()
{
   cout << "I am now inside the function second.\n";
}
```

Program Output:

Program Output

(The program's output is the same as the output of Program 6-3.)

Sending Data into a Function:

When a function is called, the program may send values into the function

```
// This program demonstrates a function with a parameter.
#include <iostream>
using namespace std;

// Function Prototype
void displayValue(int);

int main()
{
   cout << "I am passing 5 to displayValue.\n";
   displayValue(5);  // Call displayValue with argument 5
   cout << "Now I am back in main.\n";
   return 0;
}
```

```
//*********************************************************
// Definition of function displayValue.                   *
// It uses an integer parameter whose value is displayed. *
//*********************************************************

void displayValue(int num)
{
   cout << "The value is " << num << endl;
}
```

Program Output:

```
Program Output
I am passing 5 to displayValue.
The value is 5
Now I am back in main.
```

Value Returning Functions:

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:

Example:

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

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

// Outputs 8 (5 + 3)

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

Example:

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

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

Local and Global Variables:

Global variables are declared outside any function, and they can be accessed (used) on any function in the program. Local variables are declared inside a function, and can be used only inside that function. It is possible to have local variables with the same name in different functions. Even the name is the same, they are not the same. It's like two people with the same name. Even the name is the same, the persons are not.

The scope of a variable refers to where a variable is visible or accessible. If a person asks what the scope of a variable is, she's asking whether it is local or global.

```
// What is a global variable? A local variable? Scope?

// Global variable
float a = 1;

void my_test() {
  // Local variable called b.
  // This variable can't be accessed in other functions
  float b = 77;
  println(a);
  println(b);
}

void setup() {
  // Local variable called b.
  // This variable can't be accessed in other functions.
  float b = 2;
  println(a);
  println(b);
  my_test();
  println(b);
}

// You can have local variables with the same name in different functions.
// It's like having two persons with the same name: they are different persons!
```

Parameter	Local	Global
Scope	It is declared inside a function.	It is declared outside the function.
Value	If it is not initialized, a garbage value is stored	If it is not initialized zero is stored as default.
Lifetime	It is created when the function starts execution and lost when the functions terminate.	It is created before the program's global execution starts and lost when the program terminates.
Data sharing	Data sharing is not possible as data of the local variable can be accessed by only one function.	Data sharing is possible as multiple functions can access the same global variable.
Parameters	Parameters passing is required for local variables to access the value in other function	Parameters passing is not necessary for a global variable as it is visible throughout the program
Modification of variable value	When the value of the local variable is modified in one function, the changes are not visible in another function.	When the value of the global variable is modified in one function changes are visible in the rest of the program.
Accessed by	Local variables can be accessed with the help of statements, inside a function in which they are declared.	You can access global variables by any statement in the program.
Memory storage	It is stored on the stack unless specified.	It is stored on a fixed location decided by the compiler.

Advantages of using Global variables

You can access the global variable from all the functions or modules in a program

You only require declaring global variable single time outside the modules.

It is ideally used for storing "constants" as it helps you keep the consistency.

A Global variable is useful when multiple functions are accessing the same data.

Advantages of using Local Variables

The use of local variables offer a guarantee that the values of variables will remain intact while the task is running

If several tasks change a single variable that is running simultaneously, then the result may be unpredictable. But declaring it as local variable solves this issue as each task will create its own instance of the local variable.

You can give local variables the same name in different functions because they are only recognized by the function they are declared in.

Local variables are deleted as soon as any function is over and release the memory space which it occupies.

Disadvantages of using Global Variables

Too many variables declared as global, and then they remain in the memory till program execution is completed. This can cause of Out of Memory issue.

Data can be modified by any function. Any statement written in the program can change the value of the global variable. This may give unpredictable results in multi-tasking environments.

Static Variables:

Static variables in a Function: When a variable is declared as static, space for it gets allocated for the lifetime of the program. Even if the function is called multiple times, space for the static variable is allocated only once and the value of variable in the previous call

gets carried through the next function call. This is useful for implementing coroutines in C/C++ or any other application where previous state of function needs to be stored.

```
// C++ program to demonstrate
// the use of static Static
// variables in a Function
#include <iostream>
#include <string>
using namespace std;

void demo()
{
    // static variable
    static int count = 0;
    cout << count << " ";

    // value is updated and
    // will be carried to next
    // function calls
    count++;
}

int main()
{
    for (int i=0; i<5; i++)
        demo();
    return 0;
}
```

Output:

```
0 1 2 3 4
```

Lab Tasks:

Task 1

Write a C++ Program that contains four user defined function(s): addition(), subtraction(), division(), multiplication(). Develop a calculator as follows:

In main() function:

- A menu with choices addition, subtraction, division and multiplication must be displayed.
- Get two numbers and a choice from user
- Call the respective functions with user given number as parameter using switch statement
- Print the result from addition (), subtraction (), division (), multiplication().

In user defined functions:

-Plus and minus function get two integer values and return integer.
-Multiply and Divide functions get two integer values and return float.

Task2:

Write a C++ Program that contains one user defined function cal_grades().

In main() function:

-Prompt user to enter obtained (0 - 100) marks for one subject.
-Call cal_grades(marks_subject).
-Print the corresponding Grade with respect to Marks.

In user defined function:

-Perform conditioning with else if statement return char value.
-Function must return value.

Task3:

Write a program that asks the user to enter an item's wholesale cost and its markup percentage. It should then display the item's retail price.

For example:

• If an item's wholesale cost is 5.00 and its markup percentage is 100%, then the item's retail price is 10.00.

If an item's wholesale cost is 5.00 and its markup percentage is 50%, then the item's retail price is 7.50.

The program should have a function named calculateRetail that receives the wholesale cost and the markup percentage as arguments and returns the retail price of the item.

You can use this formula:

retailPrice = (wholesaleCost * markupPercent) + wholesaleCost

Input Validation: Do not accept negative values for either the wholesale cost of the item or the markup percentage.