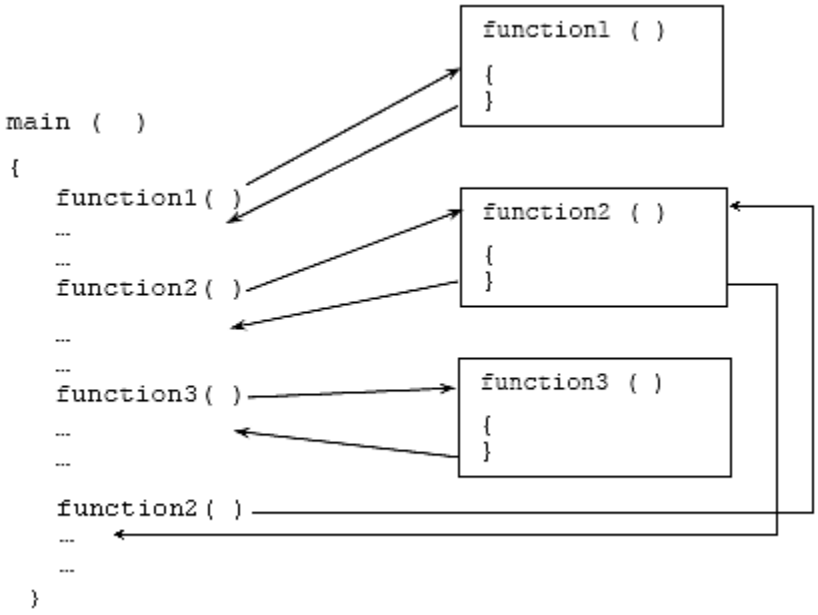
Computer Programming

Lab Manual No 07

Functions in C++



Semester:

Spring 2016

Objectives:-

The objectives of this session is:

- Learn about Functions.
- Examine various operations on a Functions.
- Function and Parameter Declarations.

Introduction:-

Professional programs are designed, coded, and tested much like hardware: as a set of modules integrated to perform a completed whole. A good analogy is an automobile; one major module is the engine, another is the transmission, a third the braking system, a fourth the body, and so on. All these modules are linked together and placed under the driver's control, which can be compared to a main () program module. The whole now operates as a complete unit, able to do useful work, such as driving to the store. During the assembly process, each module is constructed, tested, and found to be free of defects (bugs) before it's installed in the final product.

In this analogy, each major car component can be compared to a function. For example, the driver calls on the engine when the gas pedal is pressed. The engine accepts inputs of fuel, air, and electricity to turn the driver's request into a useful product—power—and then sends this output to the transmission for further processing. The transmission receives the engine's output and converts it to a form the wheels can use. An additional input to the transmission is the driver's selection of gears (drive, reverse, neutral, and so on).

The engine, transmission, and other modules "know" only the universe bounded by their inputs and outputs. The driver doesn't need to know the internal operation of the modules being controlled. All that's required is knowing what each module does and how to "call" on it when the module's output is needed. Communication between modules is restricted to passing inputs to each module as it's called on to perform its task, and each module operates in a fairly independent manner. Programmers use this same modular approach to create and maintain reliable C++ programs by using functions.

As you have seen, each C++ program must contain a main ()function. In addition to this required

function, C++ programs can also contain any number of other functions. In this session, you learn how to write these functions, pass data to them, process the passed data, and return a result.

Function and Parameter Declarations:-

In creating C++ functions, you must be concerned with the function itself and how it interacts with other functions, such as main(). Interaction with a function includes passing data to a function correctly when it's called (inputs) and returning values from a function (outputs) when it ceases operation. This section describes the first part of the interface, passing data to a function and having the function receive, store, and process the transmitted data correctly.

As you have already seen with mathematical functions, a function is called, or used, by giving the function's name and passing any data to it, as arguments, in the parentheses following the function name (see Figure 1). The called function must be able to accept the data passed to it by the function doing the calling. Only after the called function receives the data successfully can the data be manipulated to produce a useful result.

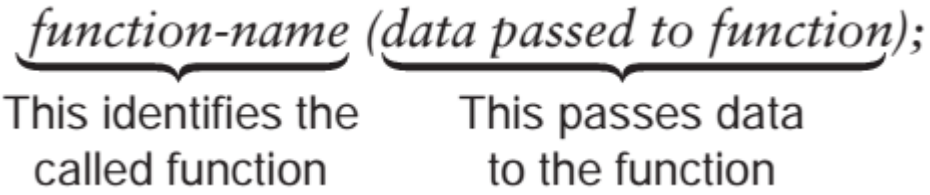


Figure 1

To clarify the process of sending and receiving data, take a look at Program 1, which calls a function named findMax(). The program, as shown, is not yet complete. After the function findMax() is written and included in Program 1, the completed program, consisting of the functions main() and findMax(), can be compiled and run.

Program 1:-

```
#include<iostream>

Using namespace std;

voidfindMax(int,int);     //the function declaration(prototype)

int main()

{

int firstnum, secnum;

cout<<"\n Enter a number:";
```

```
cin>>firstnum;

cout<<"Great! Please enter a second number:";

cin>>secnum;

findMax(firstnum,secnum);      //the function is called here

return 0;

}
```

First, examine the declaration and calling of the **findMax()** function from main(). You then see how to write **findMax()** to accept data passed to it, determine the largest or maximum value of the two passed values, and display the maximum value.

The **findMax()** function is referred to as the called function because it's called or summoned into action by its reference in main(). The function that does the calling—in this case, main ()—is referred to as the calling function. The terms "called" and "calling" come from standard phone usage, in which one party calls the other: The person initiating the call is the calling party, and the person receiving the call is the called party. The same terms describe function calls. The called function—in this case, **findMax()**—is declared as a function that expects to receive two integer numbers and to return no value (a void) to main(). This declaration is formally called a function prototype. The function is then called by the last statement in the program.

Function Prototypes:-

Before a function can be called, it must be declared to the function that will do the calling. The declaration statement for a function is referred to as a function prototype. The function prototype tells the calling function the type of value that will be formally returned, if any, and the data type and order of the values the calling function should transmit to the called function. For example, the function prototype used in Program 1.

```
void findMax(int , int);
```

declares that the **findMax()** function expects two integer values to be sent to it and returns no value (void).

Function prototypes can be placed with the variable declaration statements of the calling function, above the calling function name, as in Program 1, or in a separate header file specified

with a **#include** preprocessor statement. The function prototype for **findMax()** could have been placed before or after the statement **#include<iostream>,** before main(), or within main().

The general form of function prototype statements is as follows:

```
returnDataType functionName(list of argument data types);
```

The **returnDataType** refers to the type of value the function returns. Here are some examples of function prototypes:

```
int fmax(int,int);

double swap(int, char, char, double);

void display(double, double);
```

The function prototype for **fmax()** declares that this function expects to receive two integer arguments and returns an integer value. The function prototype for **swap()** declares that this function requires four arguments—consisting of an integer, two characters, and a double-precision argument, in that order—and returns a double-precision number. Finally, the function prototype for **display()** declares that this function requires two double-precision arguments and doesn't return any value. This function might be used to display the results of a computation without returning any value to the called function.

Using function prototypes permits the compiler to error-check data types. If the function prototype doesn't agree with data types defined when the function is written, a warning message is displayed when the program is compiled. The prototype also serves another task: It ensures that all arguments passed to the function are converted to the declared data type when the function is called.

Calling a Function:-

Calling a function is rather easy. The only requirements are using the name of the function and enclosing any data passed to the function in the parentheses following the function name, using the same order and type declared in the function prototype. The items enclosed in parentheses are called arguments of the called function (see Figure 2).

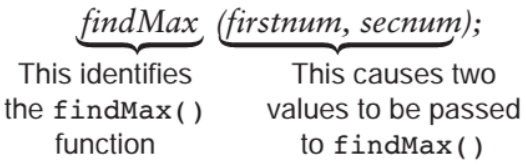


Figure 2

If a variable is one of the arguments in a function call, the called function receives a copy of the value stored in the variable. For example, the statement findMax(firstnum , secnum);calls the **findMax()** function and causes the values stored in the variables firstnum and secnum to be passed to **findMax().** The variable names in parentheses are arguments that provide values to the called function. After values are passed, control is transferred to the called function.

As shown in Figure 3, the **findMax()** function does not receive the variables named firstnum and secnum and has no knowledge of these variable names. The function simply receives the values in these variables and must then determine where to store these values before it does anything else. Although this procedure for passing data to a function might seem surprising, it's actually a safety measure for ensuring that a called function doesn't inadvertently change data stored in a variable. The function gets a copy of the data to use. It can change its copy and, of course, change any variables declared inside it. However, unless specific steps to do so are taken, a function isn't allowed to change the contents of variables declared in other functions.

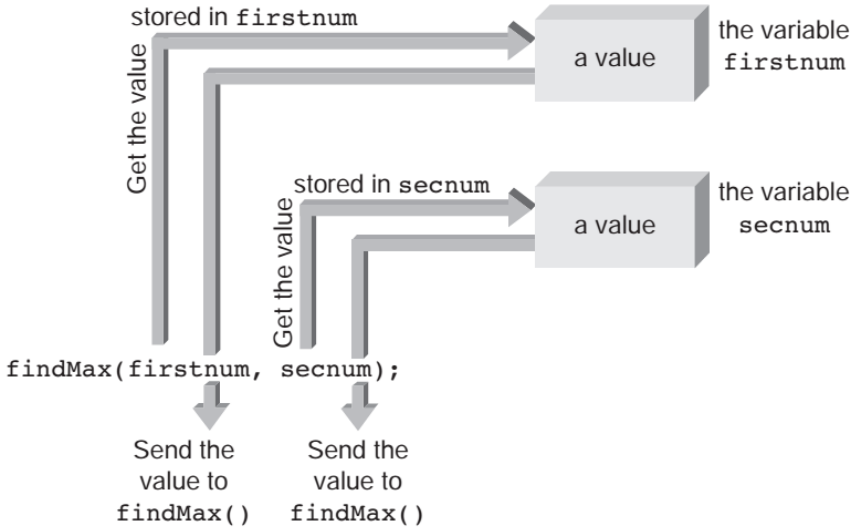


Figure 3

Next, you begin writing the **findMax()** function to process the values passed to it.

Defining a Function:-

A function is defined when it's written. Each function is defined once (that is, written once) in a program and can then be used by any other function in the program that declares it suitably. Like the main() function, every C++ function consists of two parts, a function header and a function body, as shown in Figure 4. The function header's purpose is to identify the data type of the value the function returns, give the function a name, and specify the number, order, and type of arguments the function expects. The function body's purpose is to operate on the passed data and return, at most, one value directly back to the calling function.

The function header is always the first line of a function and contains the function's return value type, its name, and the names and data types of its arguments. Because findMax()doesn't formally return any value and receives two integer values, the following function header can be used:

```
void findMax(int x, int y)
```

no semicolon

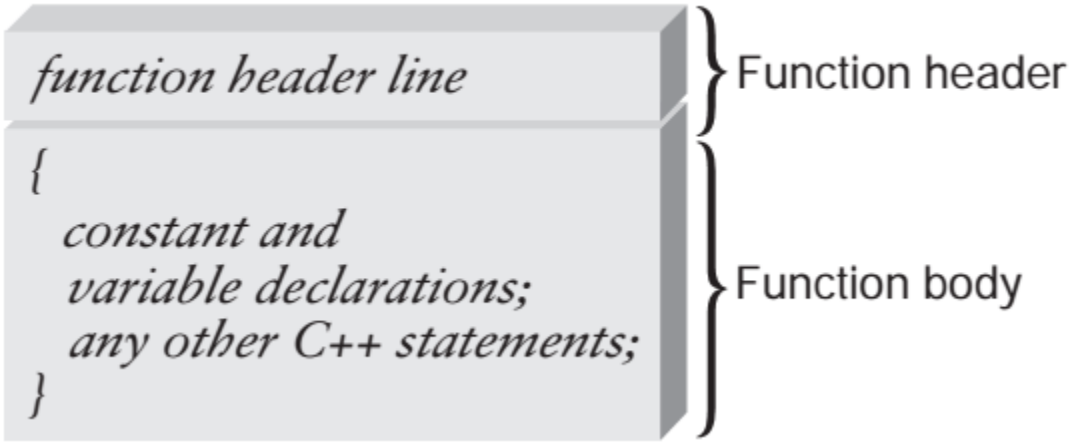


Figure 4

The argument names in parentheses in the header are called the formal parameters of the function (or parameters, for short). Therefore, the parameter x is used to store the first value passed to findMax() and the parameter y is used to store the second value passed at the time of the function call. The function doesn't know where the values come from when the call is made

from main(). The first part of the call procedure the computer executes involves going to the variables firstnum and secnum and retrieving the stored values. These values are then passed to findMax() and stored in the parameters x and y (see Figure 5).

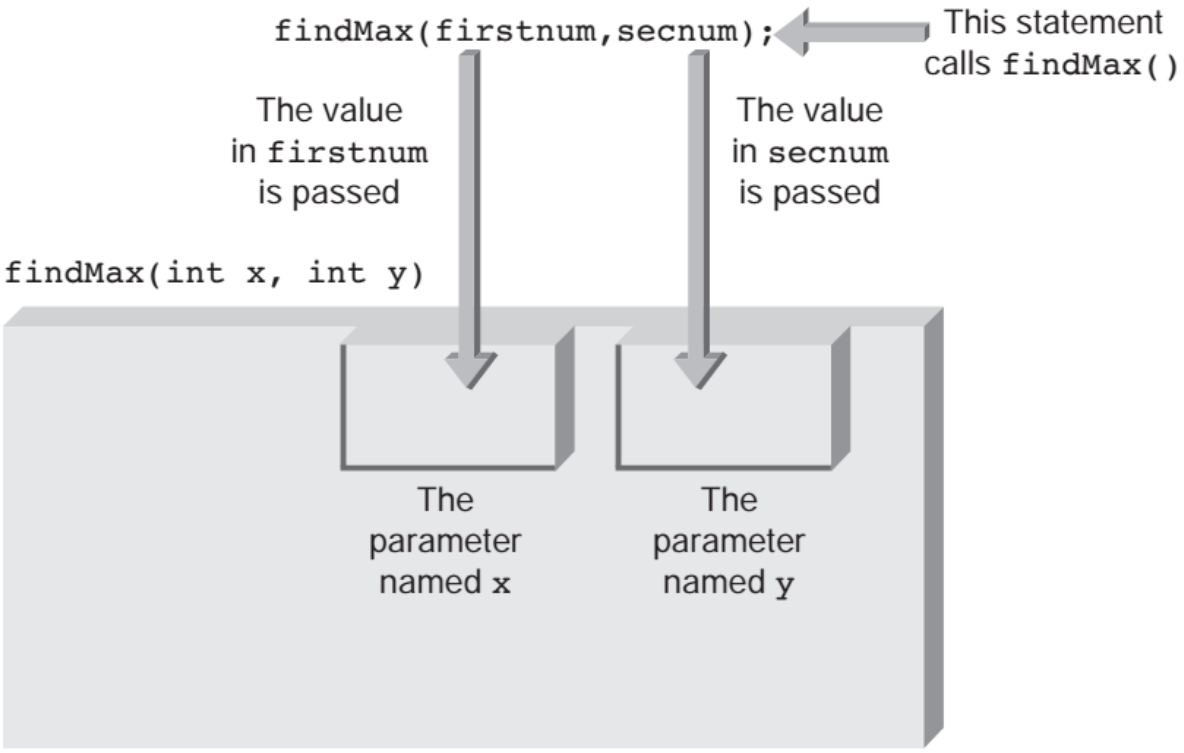


Figure 5

The function name and all parameter names in the header—in this case, findMax, x, and y—are chosen by the programmer. Any names selected according to the rules for choosing variable names can be used. Each parameter listed in the function header must include a data type. If more than one parameter is declared in the function header, they must be separated by commas and have their data types declared separately.

Summary:-

When you write a function, you're formally creating a function definition. Each definition begins with a header line that includes a parameter list, if any, enclosed in parentheses and ends with the closing brace that terminates the function's body. The parentheses are required whether or not the function uses any parameters. The following is a commonly used syntax for a function definition:

```
returnDataType functionName(parameter list)
{
    constant declarations
    variable declarations

  other C++ statements

  return value
}
```

As you've learned, a function prototype declares a function. The syntax for a function prototype, which provides the function's return data type, the function's name, and the function's parameter list, is as follows:

```
returnDataType functionName(list of parameter data types);
```

Generally, all function prototypes are placed at the top of the program, and all definitions are placed after the main()function. However, this placement can be changed. The only requirement in C++ is that a function can't be called before it has been declared or defined.

Now that the function header for findMax() has been written, you can construct its body.

The function is to select and display the larger of the two numbers passed to it.

A function body begins with an opening brace, {, contains any necessary declarations and other C++ statements, and ends with a closing brace, }. This structure should be familiar because it's the same one used in all the main() functions you've seen so far. This required structure shouldn't be a surprise because main() is a function and must adhere to the rules for constructing all legitimate functions, as shown here:

```
{
  variable declarations and
  other C++ statements
}
```

In the body of the findMax() function, one variable is declared to store the maximum of the two numbers passed to it. An if-else statement is then used to find the maximum of the two numbers. Finally, a cout statement is used to display the maximum. The following shows the complete function definition for findMax():

```
void findMax() (int x, int y)
{                    // start of function body
  int maxnum;        // variable declaration

  if (x >= y)        // find the maximum number
    maxnum = x;
  else
    maxnum = y;

  cout << "\nThe maximum of the two numbers is "
       << maxnum << endl;

  return;
}  // end of function body and end of function
```

Notice that the parameter declarations are made in the function header, and the variable declaration is made immediately after the function body's opening brace. This placement is in keeping with the concept that parameter values are passed to a function from outside the function, and variables are declared and assigned values from inside the function body.

Program 2 includes the findMax() function in the program code previously listed in Program 1.

Program 2:-

```
#include <iostream>
using namespace std;
void findMax(int, int);       // the function prototype

int main()
{
  int firstnum, secnum;

  cout << "\nEnter a number: ";
  cin  >> firstnum;
  cout << "Great! Please enter a second number: ";
  cin  >> secnum;

  findMax(firstnum, secnum);  // the function is called here

  return 0;
}

// following is the function findMax()

void findMax(int x, int y)
{                   // start of function body
  int maxnum;       // variable declaration

  if (x >= y)       // find the maximum number
    maxnum = x;
  else
    maxnum = y;

  cout << "\nThe maximum of the two numbers is "
       << maxnum << endl;

  return;
}    // end of function body and end of function
```

The placement of the findMax() function after the main() function in Program 2 is a matter of choice. Usually, main() is listed first because it's the driver function that gives anyone reading the program an idea of what the complete program is about before encountering the details of each function. In no case, however, can the definition of findMax() be placed inside main(). This rule applies to all C++ functions, which must be defined by themselves outside any other function. Each C++ function is a separate and independent entity with its own parameters and variables; nesting functions is never permitted.

Placement of Statements:-

C++ doesn't impose a rigid statement-ordering structure on programmers. The general rule for placing statements in a C++ program is simply that all preprocessor directives, symbolic constants, variables, and functions must be declared or defined before they can be used. As noted previously, although this rule permits placing both preprocessor directives and declaration statements throughout a program, doing so results in poor program structure.

As a matter of good programming form, the following statement ordering should form the basic structure around which all C++ programs are constructed:

```
preprocessor directives
function prototypes

  int main()
  {
    // symbolic constants
    // variable declarations

    // other executable statements

    // return statement
  }

  // function definitions
```

Example:-

Write a program to pass two variables as arguments to the function to calculate their sum and print them on the screen.

```
#include<iostream>

#include <cstdlib>

using namespace std;

void sum(int,int);

int main()

{

    int a ,b;

    cout<<"Enter the first number\n";

    cin>>a;

    cout<<"Enter the second number\n";

    cin>>b;

    sum(a,b);

    cout<<"I have called the sum function to execute the program\n";

    system("pause");

    return 0;

}

void sum(int x,int y)

{

     int sum;
```

```
    sum=x+y;

    cout<<"The sum of two numbers is "<<sum<<"\n";

}
```

Lab Task:-

1. Write a program to pass a number as arguments to the function body. Print the table of that number in the function body.
2. Write a program for calculator by passing three arguments to the function body, first for numeric number, 2nd for arithmetic operator and third for other numeric numbers. Perform arithmetic operation on the given number and print the result on the screen.
3. Write a program to pass two numbers as arguments to the function body compare the numbers in the function body and print the greater number on screen.

Passing Arrays as Arguments to Functions:-

An array variable can also be passed to a function as an argument. When an array is passed to the function, only the starting address of the array is passed to the function. The C++ does not make a separate copy of the array in the body of the function. It only assigns the starting address of the same memory area of the array to the array name used in the declarator of the function.

The function can change the contents of the array by directly accessing the memory cell where the array elements are stored. Thus although the starting address of the array is passed, the elements of the array can be changed as if they have been passed to the function.

Declaration of a Function with Array Arguments:-

When an array variable is used as an argument of the function, it is mentioned in the function declaration.

For example if a function is to use two arrays as arguments, one of float type one dimensional array and the other of integer type on dimensional array, it is declared as:

```
void max(float[] , int[]);
```

If two dimensional array are to be used as arguments, two square brackets are used as shown below:

```
void temp(float[][], int[][]);
```

Function Definition with Array Arguments:-

The name of the array and its type is given in declaratory of the function definition. The type of the array in declaratory must correspond with its type given in the prototype of the function.

For example the function definition with array arguments is written as:

```
void max(float xy[] , int ab[])
{
     Body of the function
}
```

The size of the array can be given both in the function declaration and in the function definition but its use is optional.

Calling a function with array arguments:-

When a function that uses an array as argument is called then only the name of the array without subscript is given. This is because only memory address of the array is passed to the function. Data is accessed from the same memory location.

The name of the array used in the function definition and in the function call may be different or same. Because it is the name of the array in the function definition to which the address of the array is passed.

Example:-

Write a program to find a number and its location in the array using a function.

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

```
void find(int[],int);
int main()
{
    int a[10] ,i,n;
    cout<<"Enter values into the array ? \n";
    for(i=0;i<=9;i++)
    cin>>a[i];
    cout<<"Enter number to find\n";
    cin>>n;
    find(a,n);
    system("pause");
    return 0;
}
void find(int x[],int a)
{
     int p=0,count=0;
     for(int c=0;c<=9;c++)
     if(a==x[c])
     {
     p=c;
     count++;
     }
     if(count==0)
```

```
        cout<<"Number not found\n";

        else

        cout<<"Number found at "<<p+1<<" position"<<endl;

}
```

Lab Task:-

1. Write a program to find the maximum number from the array by using functions.
2. Write a program to pass a string to a function and then to find out its length.
3. Write a program in C++ to pass two strings to a function and copy the second string into the first string.