



Object Oriented Programming (CS02203)

Lab Report

Name: Noor Baho
Registration #: CSU-F12-116
Lab Report #: 04
Dated: 21-11-2018
Submitted To: Mr. Usman Ahmed

The University of Lahore, Islamabad Campus
Department of Computer Science & Information Technology

Experiment # 4

Functions and Function overloading in C++

Objective

The objective of this lab is to get the understanding of function, types of functions and define user functions and return value from function.

Software Tool

- OS Windows 10
- Editor DEV C++
- Language C++

1 Theory

1.1 Function

A function is a subprogram that acts on data and often returns a value. A program written with numerous functions is easier to maintain, update and debug than one very long program. By programming in a modular (functional) fashion, several programmers can work independently on separate functions which can be assembled at a later date to create the entire project. Each function has its own name. When that name is encountered in a program, the execution of the program branches to the body of that function. When the function is finished, execution returns to the area of the program code from which it was called, and the program continues on to the next line of code.

1.2 Creating user-defined functions

Declare the function

The declaration, called the FUNCTION PROTOTYPE, informs the compiler about the functions to be used in a program, the argument they take and the type of value they return.

Define the function The function definition tells the compiler what task the function will be performing. The function prototype and the function definition must be same on the return type, the name, and the parameters. The only difference between the function prototype and the function header is a semicolon. The function definition consists of the function header and its body. The header is EXACTLY like the function prototype, EXCEPT that it contains NO terminating semicolon.

Prototyping, defining and calling a function

```
#include <iostream>
using namespace std;
void starline();           // prototype the function
int main()
{
    starline( );           // function call
    cout<< "\t\tBjarne Stroustrup\n";
    starline( );           // function call
    return 0;
}
// function definition
void starline()
{
    int count; // declaring a LOCAL variable
    for(count = 1; count <=65; count++)
        cout<< "*";
    cout<<endl;
}
```

1.3 Argument to a function

Sometimes the calling function supplies some values to the called function. These are known as parameters. The variables which supply the values to a calling function called actual parameters. The variable which receive the value from called statement are termed formal parameters. Consider the following example that evaluates the area of a circle.

```
#include<iostream>
```

```

using namespace std;
void area(float );
int main()
{
    float radius;
    cin>>radius;
    area(radius);
    return 0;
}
void area(float r)
{

}

```

Here radius is called actual parameter and r is called formal parameter.

Return type of a function

Example program

```

#include <iostream>
using namespace std;
int timesTwo(int num); // function prototype
int main()
{
    int number, response;
    cout<<"Please enter a number:";
    cin>>number;
    response = timesTwo(number); //function call
    cout<< "The answer is "<<response;
    return 0;
}
//timesTwo function
int timesTwo (int num)
{
    int answer; //local variable
    answer = 2 * num;
    return (answer);
}

```

1.4 Calling of a function

The function can be called using either of the following methods:

- Call by Value**

- Call by Reference**

Call by value

In call by value method, the called function creates its own copies of original values sent to it. Any changes, that are made, occur on the function's copy of values and are not reflected back to the calling function.

Call by reference

In call by reference method, the called function accesses and works with the original values using their references. Any changes, that occur, take place on the original values are reflected back to the calling code. Consider the following program which will swap the value of two variables.

Using Call by Reference

```
#include<iostream>
using namespace std;
void swap(int &, int &);
int main()
{
    int a=10,b=20;
    swap(a,b);
    cout<<a<<" "<<b;
    return 0;
}
void swap(int &c, int &d)
{
    int t;
    t=c;
    c=d;
    d=t;
}
```

output:

20 10

Using Call by Reference

```
#include<iostream>
using namespace std;
void swap(int , int );
int main()
{
    int a=10,b=20;
    swap(a,b);
    cout<<a<<" "<<b;
    return 0;
}
void swap(int c, int d)
{
    int t;
    t=c;
    c=d;
    d=t;
}
```

output:

10 20

Function with default arguments

C++ allows to call a function without specifying all its arguments. In such cases, the function assigns a default value to a parameter which does not have a matching arguments in the function call. Default values are specified when the function is declared. The compiler knows from the prototype how many arguments a function uses for calling. Example

```
float result(int marks1, int marks2, int marks3=75);
```

a subsequent function call

```
average = result(60,70);
```

passes the value 60 to marks1, 70 to marks2 and lets the function use default value of 75 for marks3.

The function call

```
average = result (60,70,80);
```

passes the value 80 to marks3

.

1.5 Inline function

Functions save memory space because all the calls to the function cause the same code to be executed. The functions body need not be duplicated in memory. When the compiler sees a function call, it normally jumps to the function. At the end of the function. it normally jumps back to the statement following the call. While the sequence of events may save memory space, it takes some extra time. To save execution time in short functions, inline function is used. Each time there is a function call, the actual code from the function is inserted instead of a jump to the function. The inline function is used only for shorter code.

```
Inline int cube (int r)
{
    return r*r*r;
}
```

Some important points to be noted

- Function is made inline by putting a word inline in the beginning.
- Inline function should be declared before main () function.
- It does not have function prototype.
- Only shorter code is used in inline function if longer code is made inline then compiler

1.6 Global variable and local variable

Local variable: A variable declared within the body of a function will be evaluated only within the function. The portion of the program in which a variable is retained in memory is known as the scope of the variable. The scope of the local variable is a function where it is defined. A variable may be local to function or compound statement.

Global variable: a variable that is declared outside any function is known

as a global variable. The scope of such a variable extends till the end of the program. These variables are available to all functions which follow their declaration. So it should be defined at the beginning, before any function is defined.

Unary scope resolution operator (::) It is possible to declare local and global variables of the same name. C++ provides the unary scope resolution operator (::) to access a global variable when a local variable of the same name is in scope. A global variable can be accessed directly without the unary scope resolution operator if the name of the global variable is not the same as that of a local variable in scope.

1.7 Function overloading

Function overloading allows you to use the same name for different functions. Function overloading is usually used to enhance the readability of the program. If you have to perform one single operation but with different number or types of arguments, then you can simply overload the function.

Ways to overload a function

- By changing number of Arguments.
- By having different types of argument.

Different number of arguments

This type of function overloading we define two functions with same names but different number of parameters of the same type. For example, in the below mentioned program we have made two sum() functions to return sum of two and three integers.

```
int sum (int x, int y)
{
    cout << x+y;
}
int sum(int x, int y, int z)
{
    cout << x+y+z;
}
```


Here sum() function is overloaded, to have two and three arguments. Which sum() function will be called, depends on the number of arguments.

```
int main()
{
    sum (10,20); // sum() with 2 parameter will be called
    sum(10,20,30); //sum() with 3 parameter will be called
}
```

Different data type of arguments

In this type of overloading we define two or more functions with same name and same number of parameters, but the type of parameter is different. For example in this program, we have two sum() function, first one gets two integer arguments and second one gets two double arguments.

```
int sum(int x,int y)
{
    cout<< x+y;
}
double sum(double x,double y)
{
    cout << x+y;
}
int main()
{
    sum (10,20);
    sum(10.5,20.5);
}
```

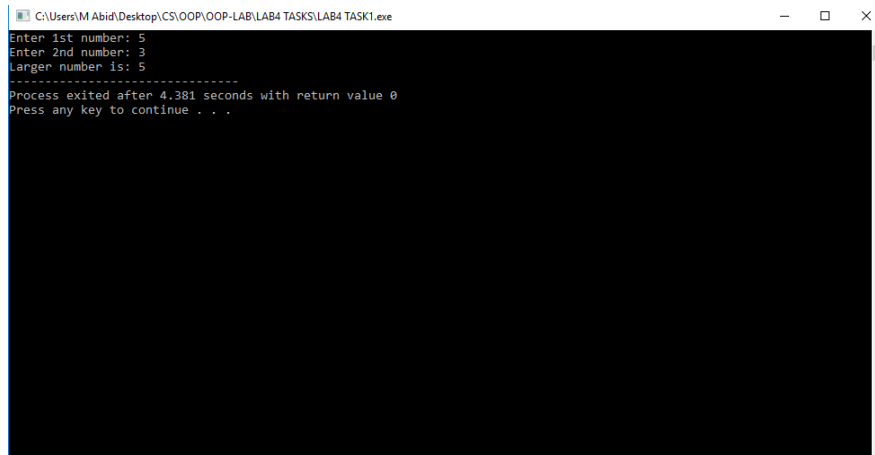


Figure 1: Showing Larger Number

2 Task

2.1 Procedure: Task 1

Write a function that takes two integers and returns the larger one.

Solution:

```
#include<iostream>
//task1 function that take two integer and return larger
using namespace std;
int func(int x, int y)
{
    if(x> y)
    {
        return x;
    }else if(y>x)
    {
        return y;
    }
    else
    cout<<"Both are equal.";
}
```

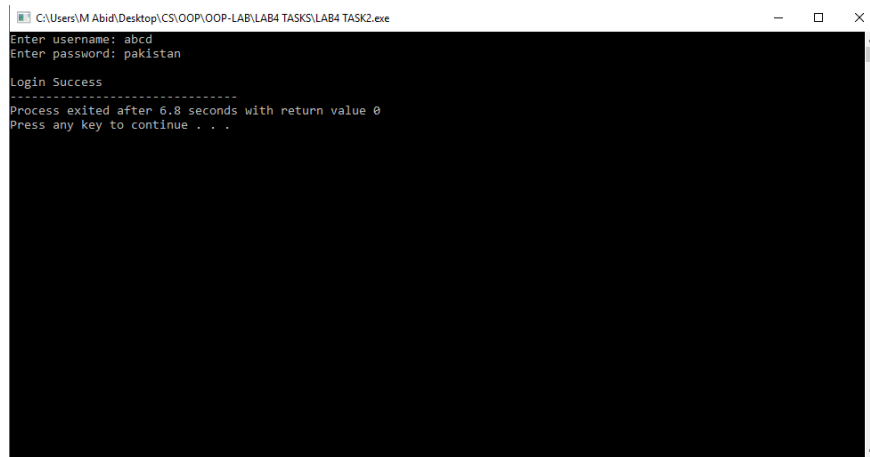
```

int main()
{
    int a, b;
    cout<<"Enter 1st number: ";
    cin>>a;
    cout<<"Enter 2nd number: ";
    cin>>b;
    cout<<"Larger number is: "<<func(a,b);

    return 0;
}

```

Explanation: Greater number was required to be identified so We created a function with two parameters, taking two numbers and retruning the number which is greater after comparison.

A screenshot of a Windows command prompt window titled "C:\Users\M Abid\Desktop\CS\OOP\OOP-LAB\LAB4 TASKS\LAB4 TASK2.exe". The window has a black background with white text. The text shows the program's execution: "Enter username: abcd", "Enter password: pakistan", "Login Success", "-----", "Process exited after 6.8 seconds with return value 0", and "Press any key to continue . . .". The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
C:\Users\M Abid\Desktop\CS\OOP\OOP-LAB\LAB4 TASKS\LAB4 TASK2.exe
Enter username: abcd
Enter password: pakistan
Login Success
-----
Process exited after 6.8 seconds with return value 0
Press any key to continue . . .
```

Figure 2: Login Successful

2.2 Procedure: Task 2

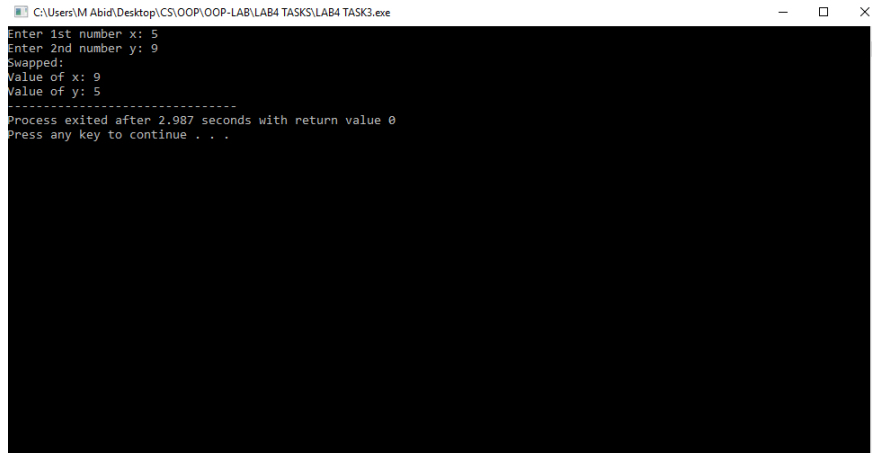
Write a function that takes username and password as string and display message "login in success" if username is "abcd" and password is="pakistan" and display message "Login failed" otherwise.

Solution:

```
#include<iostream>
#include<string>
//task2 login successfull
using namespace std;
void verify(string user , string pswd)
{
    if (user=="abcd" && pswd=="pakistan")
    {
        cout<<"Login Success ";
    }
    else
        cout<<"Login failed ";
}
int main()
{
    string a,b;
```

```
        cout<<"Enter  username:  ";  
        cin>>a;  
        cout<<"Enter  password:  ";  
        cin>>b;  
        cout<<endl;  
        verify(a,b);  
        return 0;  
    }
```

Explanation: We just took username and password from user, then passed as parameter to a function which is comparing both strings with the default username and password using & operator to make sure both conditions are followed and then executing accordingly.



```
C:\Users\IM Abid\Desktop\CS\OOP\LAB\LAB4 TASKS\LAB4 TASK3.exe
Enter 1st number x: 5
Enter 2nd number y: 9
Swapped:
Value of x: 9
Value of y: 5
-----
Process exited after 2.987 seconds with return value 0
Press any key to continue . . .
```

Figure 3: Two Numbers Swapped

2.3 Procedure: Task 3

Write a function that swaps two numbers void swap() using reference parameters.

Solution:

```
#include<iostream>
//task3 swap integers by reference
using namespace std;
void swapnum(int &x, int &y)
{
    int z;
    z=x;
    x=y;
    y=z;
}

int main()
{
    int a,b;
    cout<<"Enter 1st number x: ";
    cin>>a;
    cout<<"Enter 2nd number y: ";
```

```

    cin>>b;

    swapnum(a,b);
    cout<<"Swapped:"<<endl;
    cout<<"Value of x: "<<a;
    cout<<endl<<"Value of y: "<<b;
    return 0;
}

```

Explanation: Assigned two numbers to variables those are taken from user. Then in function parameters the reference of two variables are passed and proceeded to be swapped. value of x is stroing to z, y to x and then z to y. finally x and y are swapped and shown.

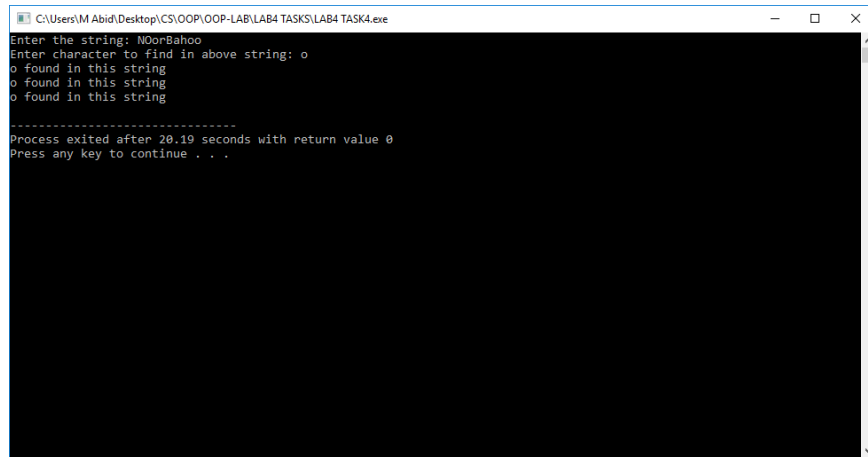


Figure 4: Finding Character in String

2.4 Procedure: Task 4

Write a C++ program with a function `int find_char(char*,char)` That finds out whether there is a given character in a string. The return value of the function `find_char` should be 1 if the given character is in the string otherwise 0.

Solution:

```
#include<iostream>
#include<conio.h>
#include<string.h>
//finding character in string
using namespace std;
int find_char(char *x, char y)
{
    if (*x==y)
    {
        return 1;
    }
    else return 0;
}

int main()
```



```

{
    string c_string;
    char character;
    char *ptr;
    int length;
    cout<<"Enter the string: ";
    cin>>c_string;
    cout<<"Enter character to find in above string: ";
    cin>>character;
    length=c_string.length();

    char ch[length];
    int result;
    for(int i=0; i<length; i++)
    {
        ch[i]=c_string[i];
        *ptr=ch[i];

        result=find_char(ptr, character);
        if(result==1)
        {
            cout<<character<<" found in this string"<<endl;
        }
    }
}

```

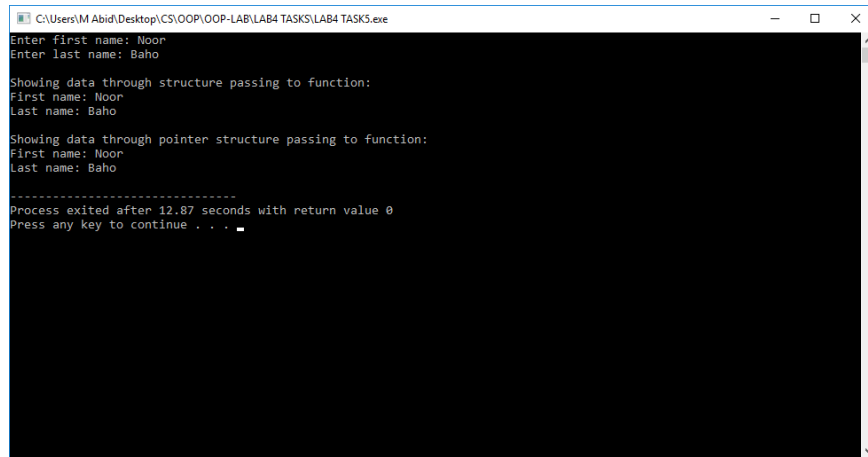
Explanation: Took a string from user and a character to be find in that string. Now defined a function fin_char which has parameters; one char pointer and other character variable. using for loop, each iteration is performing the following:

assigning string index to array index.

pointer is set to that particular index.

int variable reslut is storing the return value of function either 0 or 1 as returned by function find_char.

if statement each time checking if function returned 1 then shows the character found and if returns 0 then shows character not found in the string.

A screenshot of a Windows command prompt window titled "C:\Users\IM Abid\Desktop\CS\OOP\OOP-LAB\LAB4 TASKS\LAB4 TASK5.exe". The window shows the following text: "Enter first name: Noor", "Enter last name: Baho", "Showing data through structure passing to function:", "First name: Noor", "Last name: Baho", "Showing data through pointer structure passing to function:", "First name: Noor", "Last name: Baho", "-----", "Process exited after 12.87 seconds with return value 0", and "Press any key to continue . . .".

```
C:\Users\IM Abid\Desktop\CS\OOP\OOP-LAB\LAB4 TASKS\LAB4 TASK5.exe
Enter first name: Noor
Enter last name: Baho

Showing data through structure passing to function:
First name: Noor
Last name: Baho

Showing data through pointer structure passing to function:
First name: Noor
Last name: Baho

-----
Process exited after 12.87 seconds with return value 0
Press any key to continue . . .
```

Figure 5: Showing data by passing structure and its pointer to function

2.5 Procedure: Task 5

Define a structure of student with any data. Write two functions to display the data of structure

```
void showStudent(struct student)
void showStudent(struct student *)
```

Solution:

```
#include<iostream>
#include<conio.h>
//
using namespace std;
struct student{
    string f_name, l_name;
};
void showStudent(student a)
{
    cout<<"First name: "<<a.f_name<<endl;
    cout<<"Last name: "<<a.l_name<<endl;
}
void showStudent(student *b)
{
    cout<<"First name: "<<b->f_name<<endl;
```

```

        cout<<"Last name: "<<b->l_name<<endl;
    }
int main()
{
    student obj;
    cout<<"Enter first name: ";
    cin>>obj.f_name;
    cout<<"Enter last name: ";
    cin>>obj.l_name;

    cout<<endl<<"Showing data through structure passing to function: "<<endl;
    showStudent(obj);

    cout<<endl<<"Showing data through pointer structure passing to function: "<<endl;
    showStudent(&obj);
}

```

Explanation: Created a structure student with two members f_name and l_name. In main function obj declared of student data type. members are set by taking input from user. Now created two functions; one is taking simple structure as parameter and other is taking pointer of structure as parameter. Both access the same values and shows the user on screen.

3 Conclusion

After performing above tasks, We just learned alot about functions, function overloading etc but specially function overloading and pointer as argument at once, is the point that is cleared now. Functions overload on basis of their parameters when they are of same name. Pointer could be passed to the functions to access values which ultimately reduces complexity in the sense one pointer can be used numerous times to avoide number of variables those will occupy spaces.