**Programming Fundamentals**

**Lab Manual**

**Week 06 – Lab 02**

****

**Recursion**

**Session: Spring 2013**

**Faculty of Information Technology**

**UCP Lahore Pakistan**

# Table of Contents

[Table of Contents 2](#_Toc352956882)

[OBJECTIVES 3](#_Toc352956883)

[OVERVIEW 3](#_Toc352956884)

[Lab Task 1 5](#_Toc352956885)

[Lab Task 2 5](#_Toc352956886)

[Lab Task 3 6](#_Toc352956887)

[Lab Task 4 6](#_Toc352956888)

[Lab Task 5 **Error! Bookmark not defined.**](#_Toc352956889)

# OBJECTIVES

* Understand the concept of recursions
* Implementing various recursive functions.

# OVERVIEW

**Recursion** is when a function calls itself. That is, in the course of the function definition there is a call to that very same function. At first this may seem like a never ending loop, or like a dog chasing its tail. It can never catch it. So too it seems our function will never finish. This might be true is some cases, but in practice we can check to see if a certain condition is true and in that case exit (return from) our function. The case in which we end our recursion is called a **base case**. Additionally, just as in a loop, we must change some value and incrementally advance closer to our base case.

Consider this function.   
void myFunction( int counter)  
{  
if(counter == 0)  
     return;  
else  
       {  
       cout <<counter<<endl;  
       myFunction(--counter);  
       return;  
       }  
}

This recursion is not infinite, assuming the function is passed a positive integer value. What will the output be?   
Consider this function:

void myFunction( int counter)  
{  
if(counter == 0)  
     return;  
else  
       {  
       cout<<"hello"<<counter<<endl;  
       myFunction(--counter);  
       cout<<counter<<endl;  
       return;  
       }  
}

If the function is called with the value 4, what will the output be?

The above recursion is essentially a loop like a for loop or a while loop. When do we prefer recursion to an iterative loop? We use recursion when we can see that our problem can be reduced to a simpler problem that can be solved after further reduction.

Every recursion should have the following characteristics.

1. A simple base case which we have a solution for and a return value. Sometimes there are more than one base cases.
2. A way of getting our problem closer to the base case. I.e. a way to chop out part of the problem to get a somewhat simpler problem.
3. A recursive call which passes the simpler problem back into the function.

# Lab Task 1

Consider the following recursive function:

void exercise(int x)

{

if (x > 0 && x < 10)

{

cout << x << " ";

exercise(x + 1);

}

}

What is the output of the following statements? And draw function Call Stack.

# Lab Task 2

***(Recursive Multiplication)* MULTIPLICATION** can be achieved using multiple steps of addition

e.g

To multiply x by y  
 if x is 0  
  the answer is 0  
 if x is 1  
  the answer is y  
 otherwise  
  multiply x - 1 by y, and add y

Write a recursive function for above process and draw call stack diagram.

# Lab Task 3

***(Recursive Exponentiation)*** Write a recursive function power( base, exponent ) that when invoked returns **base exponent** For example, power( 3, 4 ) = 3 \* 3 \* 3 \* 3. Assume that exponent is an integer greater than or equal to 1. *Hint:* The recursion step would use the relationship

**base exponent = base \* baseexponent–1**

and the terminating condition occurs when exponent is equal to 1 because

**base1 = base**

# Lab Task 4

***(Recursive Exponentiation) Extension*** Modify the recursive function powerto Calculate the Power of a Number using even powers

x53=x\*(x2)26

x26=(x2)13

x13=x\*(x2)6

x6=(x2)3

x3=x\*(x2)1

x1=x\*(x2)0

# Lab Task 5

Use recursion to display a triangle of stars according to a number given by the user.

The sample output will be:

How many rows of triangle do you want to see: 7

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*

\*\*\*\*\*\*\*