Department of Software Engineering
Mehran University of Engineering and Technology, Jamshoro

Course: SWE224 – Data Structure & Algorithms			
Instructor	Mariam Memon	**Practical/Lab No.**	11
Date		**CLOs**	3
Signature		**Assessment Score**	01

Topic	Implementation of Recursion
Objectives	Program recursive functions

Lab Discussion: Theoretical concepts and Procedural steps

Recursion

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily.

A Mathematical Interpretation

Let us consider a problem that a programmer have to determine the sum of first n natural numbers, there are several ways of doing that but the simplest approach is simply add the numbers starting from 1 to n. So the function simply looks like,

approach(1) – Simply adding one by one

f(n) = 1 + 2 + 3 +……..+ n

but there is another mathematical approach of representing this,

approach(2) – Recursive adding

f(n) = 1 if n=1

f(n) = n + f(n-1) if n>1

There is a simple difference between the approach (1) and approach(2) and that is in **approach(2)** the function " **f( )** " itself is being called inside the function, so this phenomenon is named as recursion and the function containing recursion is called recursive function, at the end this is a great tool in the hand of the programmers to code some problems in a lot easier and efficient way.

What is base condition in recursion?

In the recursive program, the solution to the base case is provided and the solution of the bigger problem is expressed in terms of smaller problems.

```
int fact(int n)
{
   if (n < = 1) // base case
      return 1;
   else
      return n*fact(n-1);
}
```

In the above example, base case for n < = 1 is defined and larger value of number can be solved by converting to smaller one till base case is reached.

How a particular problem is solved using recursion?

The idea is to represent a problem in terms of one or more smaller problems, and add one or more base conditions that stop the recursion. For example, we compute factorial n if we know factorial of (n-1). The base case for factorial would be n = 0. We return 1 when n = 0.

Why Stack Overflow error occurs in recursion?

If the base case is not reached or not defined, then the stack overflow problem may arise. Let us take an example to understand this.

```
int fact(int n)
{
   // wrong base case (it may cause
   // stack overflow).
   if (n == 100)
      return 1;

   else
      return n*fact(n-1);
}
```

If fact(10) is called, it will call fact(9), fact(8), fact(7) and so on but the number will never reach 100. So, the base case is not reached. If the memory is exhausted by these functions on the stack, it will cause a stack overflow error.

Lab Tasks

Task 1: Write a program and recurrence relation to find the Fibonacci series of n where n>2 .

Task 2: Design a program for the Tower of Hanoi.

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.

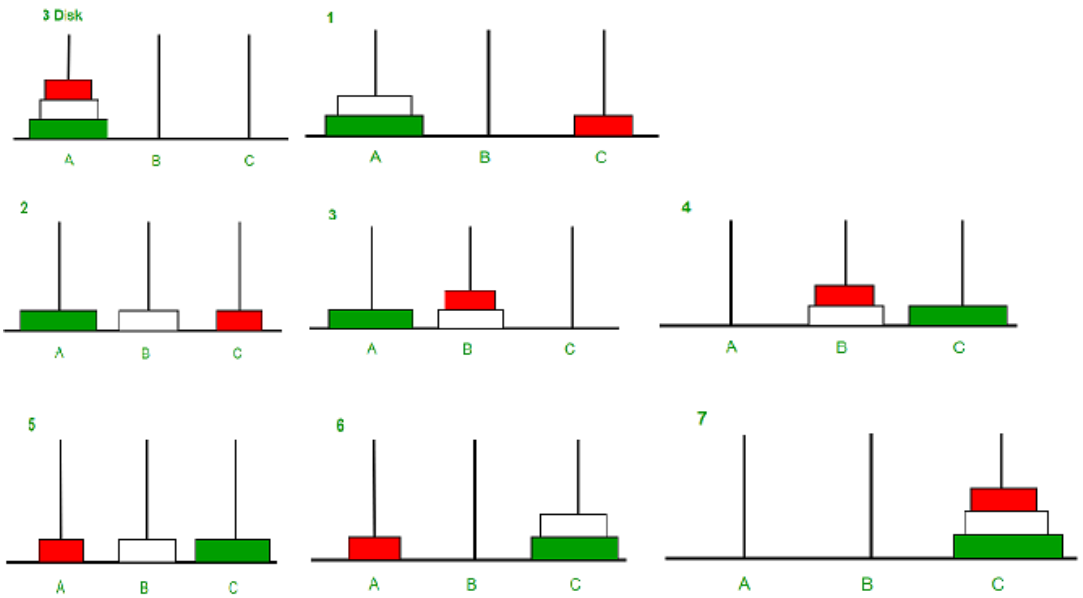
Approach :

```
Take an example for 2 disks :
Let rod 1 = 'A', rod 2 = 'B', rod 3 = 'C'.

Step 1 : Shift first disk from 'A' to 'B'.
Step 2 : Shift second disk from 'A' to 'C'.
Step 3 : Shift first disk from 'B' to 'C'.

The pattern here is :
Shift 'n-1' disks from 'A' to 'B'.
Shift last disk from 'A' to 'C'.
Shift 'n-1' disks from 'B' to 'C'.

Image illustration for 3 disks :
```



Examples:

```
Input : 2
Output : Disk 1 moved from A to B
         Disk 2 moved from A to C
         Disk 1 moved from B to C

Input : 3
Output : Disk 1 moved from A to C
         Disk 2 moved from A to B
         Disk 1 moved from C to B
         Disk 3 moved from A to C
         Disk 1 moved from B to A
         Disk 2 moved from B to C
         Disk 1 moved from A to C
```

Lab Rubrics for Evaluation

Rubrics	Proficient	Adequate	Poor
Programming Algorithms	Completely accurate and efficient design of algorithms **(0.4)**	Accurate but inefficient algorithms **(0.2)**	Unable to design algorithms for the given problems **(0.0)**
Originality	Submitted work shows large amount of original thought **(0.3)**	Submitted work shows some amount of original thought **(0.2)**	Submitted work shows no original thought **(0.0)**
Troubleshooting	Easily traced and corrected faults **(0.3)**	Traced and corrected faults with some guidance **(0.2)**	No troubleshooting skills **(0.0)**