Data Structures Lab 5

Course: Data Structures (CL2001) **Semester:** Fall 2023
Instructor: Sameer Faisal **T.A:** N/A

Note:

- Lab manual cover following below recursion topics
  {Base Condition, Direct and Indirect Recursion, Tailed Recursion, Nested Recursion, Backtracking}
- Maintain discipline during the lab.
- Just raise hand if you have any problem.
- Completing all tasks of each lab is compulsory.
- Get your lab checked at the end of the session.

Base Condition in Recursion

Sample Code

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

Key Points: 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.

Example 1: Generate the following sequence with recursive approach

1 , 3 , 6 , 10 , 15 , 21 , 28 . . . .

Task-1:

a. Generate the following sequence with recursive approach

0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55 , 89 , 144 . . .

Direct and Indirect Recursion

Sample Code (Direct Recursion)

```
void X()
{    // Some code....
   X();
   // Some code...
```

}

Sample Code (In-Direct Recursion)

```
void indirectRecFun1()
{   // Some code...
   indirectRecFun2();
   // Some code...
}
void indirectRecFun2()
{    // Some code...
   indirectRecFun1();
   // Some code...
}
```

Task-2:

a. **Write a indirect recursive code for the above task-1 (b) part with same approach as defined in the above sample code of In-Direct Recursion**

Tailed and Non Tailed Recursion

Sample Code (Non tailed Recursion)

```
unsigned int fact(unsigned int n)
{
    if (n == 0)
        return 1;

    return n * fact(n - 1);
}

// Driver program to test above function
int main()
{
    cout << fact(5);
    return 0;
}
```

Sample Code (Tailed Recursion)

```
unsigned factTR(unsigned int n, unsigned int a)//int a = accumilator
{
    if (n == 1)
        return a;

    return factTR(n - 1, n * a); //Note this is the last thing as recursive
}

// A wrapper over factTR
unsigned int fact(unsigned int n)
{
    return factTR(n, 1);
}
```

Task 3:
Sort The Unsorted Numbers with both tail recursive and Normal recursive approach
Sample Input and Output
Given array is
12 11 13 5 6 7
Sorted array is
5 6 7 11 12 13

Nested Recursion

Sample Code

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

int fun(int n)
{
    if (n > 100)
        return n - 10;

    // A recursive function passing parameter
    // as a recursive call or recursion inside
    // the recursion
    return fun(fun(n + 11));
}
int main()
{
    int r;
    r = fun(95);

    cout << " " << r;

    return 0;
}
```

Task 4:
Dry run the outputs of the upper code in order to find out how the recursive calls are made. Make a stack and visualize the functionality of stack in the case of recursion on paper.

Backtracking

Sample Pseudocode

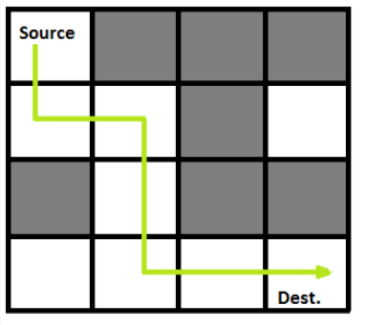
```
void findSolutions(n, other params) :
    if (found a solution) :
        solutionsFound = solutionsFound + 1;
        displaySolution();
        if (solutionsFound >= solutionTarget) :
            System.exit(0);
        return

    for (val = first to last) :
        if (isValid(val, n)) :
            applyValue(val, n);
            findSolutions(n+1, other params);
            removeValue(val, n);
```

```
boolean findSolutions(n, other params) :
    if (found a solution) :
        displaySolution();
        return true;

    for (val = first to last) :
        if (isValid(val, n)) :
            applyValue(val, n);
            if (findSolutions(n+1, other params))
                return true;
            removeValue(val, n);
        return false;
```

A Maze is given as N*N binary matrix of blocks where source block is the upper left most block i.e., maze[0][0] and destination block is lower rightmost block i.e., maze[N-1][N-1]. A rat starts from source and has to reach the destination. The rat can move only in two directions: forward and down.



In the maze matrix, 0 means the block is a dead end and 1 means the block can be used in the path from source to destination.

```
{1, 0, 0, 0}
{1, 1, 0, 1}
{0, 1, 0, 0}
{1, 1, 1, 1}
```

Following is the above-mentioned maze transformed in binary.

Task-5

A. Design the function with recursive approach to find the number of existing destination path in the above provided sample code link
B. Change the Maze with following configuration. Find the optimal path to reach the destination with recursive approach

int maze[N][N] = { { 1, 0, 0, 1 }, //Left is the source and right is the destination
{ 0, 1, 1, 1 },
{ 0, 1, 1, 0 }}