**Assignment 1: Implement a templatized Stack ADT (StackADT).**

Part A: The ADT is called **StackADT**

Required Functions:

1. bool Push (T val)//
2. bool Pop (T & newval)
3. bool IsEmpty()
4. bool IsFull()
5. bool Top(T & val)
6. StackADT() //default constructor. Creates a stack of default size say 10
7. StackADT(int size) constructor. Creates a stack of size = size

You will also need a destructor.

PartB: The main/driver program.

1. Declare an integer and character StackADT objects using the overloaded constructor.
2. Then the following code should execute without errors and WITHOUT ANY CHANGES. I will copy paste this exact code in main and if there is an error of any sort, your entire code will be considered incorrect and incomplete.

cout<<obj1.Pop(newval);

obj1.Push(1);

obj1.Push(2);

obj2.Push('3');

obj1.Push(5);

if(!obj1.IsFull())

obj1.Push(4);

for(int i=0; i<6; i++)

{

obj1.Pop(newval);

cout<<newval;

}

Part C: Application of StackADT: **Maze Solver**

1. Create a coordinate structure. It contains two integers x and y;

e.g struct coordinate {

int x,y; };

2. Read the data from file in the given format, and store it in a two dimensional character array created dynmaically. **Format of File**: no. of rows <newline> no. of columns <newline> the maze where \* is the wall and – (minus sign) is the passage.

2. Look for the opening in the maze, and then write a program that answers the following question: *Do you find a way out of the maze?*

3. Create a stack object of type coordinates. StackADT<coordinate> stack1;

4. Use the stack when ever you have more than one choice. If there is a choice (x1, y1) and (x2, y2): Store one coordinate in stack. How? Declare a coordinate object, and assign the value to it, say (x2, y2), and store this values in the stack: stack1.Push(coordinates\_object); then follow the other path..

6. If there is no way out, check the stack. If it is not empty:

Declare a coordinate object, and pop stack values into it: stack.pop(coordinate\_object). Jump to this location. And so on….

File Input: (This maze HAS a way out)

6

5

\*\*\*-\* //row [0]

\*---\* //row [1]

\*\*-\*\* //row [2]

\*\*--\* //row [3]

\*\*-\*\* //row [4]

\*\*-\*\* //row [5]

Output: Yes!