|  |  |
| --- | --- |
| Data Structures & Algorithms Diploma in IT, CSF  Year 2 (2024/25) Semester 3 | Week 4 |
| 2-3 Hours |
| **Practical 4 – Stacks** | |

**Objectives**

At the end of this practical, the students should be able to:

* Implement a Stack ADT
* Apply use of stacks in simple applications

|  |
| --- |
| **IMPORTANT**   * Upload all your work to Brightspace by the designated time stated in Brightspace. |

1. The specification of the Stack ADT implemented using Pointers is given below.

|  |
| --- |
| // Stack.h (Pointer-based implementation)  #pragma once  #include <iostream>  using namespace std;  typedef int ItemType;  class Stack  {  private:  struct Node  {  ItemType item;  Node \*next;  };  Node \*topNode;  public:  //Default constructor  Stack();  //Destructor  ~Stack();  //check if the stack is empty  bool isEmpty();  //push item on top of the stack  bool push(ItemType item);  //pop item from top of stack  bool pop();  //retrieve and pop item from top of stack  bool pop(ItemType &item);  //retrieve item from top of stack  void getTop(ItemType &item);  //display items in stack in order  void displayInOrder();  //display items in stack in order of insertion  void displayInOrderOfInsertion();  }; |

NB: for displayInOrderOfInsertion(), the original stack should still contain the original contents after execution of the function even though some manipulation of the stack may be required to display the items

2. Implement the operations of the Stack ADT

*Note : You should implement (and test) one operation at a time.*

3. Write a sample program, StackDemo.cpp, to do the following:

1. Create an empty stack, s
2. Push ‘3’ on top of the stack
3. Push ‘4’ on top of the stack
4. Get top item of the stack and display
5. Display all the items in the stack in order of insertion
6. Pop top item from stack
7. Display all the items in the stack in order of insertion

4. Write a program to keep track of visited URLs in a web browser by implementing a back button.

A skeleton program is given below:

#include <iostream>

#include <string>

#include "Stack.h"

int main()

{

Stack back\_stack;

string temp;

while (temp != "0")

{

cout << "[1] Visit URL\n[2] Back\n[0] Exit\nYour Choice: ";

cin >> temp;

if (temp == "0")

break;

}

return 0;

}

5. Add a forward button functionality to the web browser program in Q4. The forward button keeps a history of the URLs visited each time the back button is pressed. This history is cleared when a new address is entered.

Test your program with the following series of actions:

* + Enter a series of URLs (10, for example)
  + Go back to some URLs (7, for example)
  + Go forward a few, but not to the last (3, for example)
  + Enter a new URL
  + Check that the forward stack is empty by trying to go forward
  + Go back a few URLs (2, for example)

OPTIONAL

6. Write a program to display the path from a start cell to the end cell of an N x N maze. '.' in the maze means the cell can be used in the path and '#' means that the cell is a wall. The cell position is represented by the following structure, where row = 0 and col = 0 represents the top left corner of the maze:

struct Cell

{

int row;

int col;

};

The program should make use of a **stack** to enable backtracking and print the contents of the maze after each cell is chosen in the path.

The program should implement the following 2 functions:

* + Cell hasNeighbor(char [N][N], Cell);

This function takes two parameters, a maze and the current cell position and returns the next cell position to move to. If there is no valid cell position to move to, it returns the cell with row = -1 and col = -1.

* + void display(char[N][N]);

This function takes a maze as parameter and prints the maze contents.

The program is required to work with 2 arrays:

* + the original maze array.

For example:

#define N 12

char maze[N][N] =

{ '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#',

'#', '.', '.', '.', '#', '.', '.', '.', '.', '.', '.', '#',

'.', '.', '#', '.', '#', '.', '#', '#', '#', '#', '.', '#',

'#', '#', '#', '.', '#', '.', '.', '.', '.', '#', '.', '#',

'#', '.', '.', '.', '.', '#', '#', '#', '.', '#', '.', '.',

'#', '#', '#', '#', '.', '#', '.', '#', '.', '#', '.', '#',

'#', '.', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#',

'#', '#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#',

'#', '.', '.', '.', '.', '.', '.', '.', '.', '#', '.', '#',

'#', '#', '#', '#', '#', '#', '.', '#', '#', '#', '.', '#',

'#', '.', '.', '.', '.', '.', '.', '#', '.', '.', '.', '#',

'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#' };

* + a visited array. The contents of this array are '#', '.' and 'v' which represents the cells that have been visited.

Hint: Use the C++ memcpy() function to copy from the original maze array.

For more info, visit: <http://www.cplusplus.com/reference/cstring/memcpy/>