|  |  |
| --- | --- |
| **Data Structures & Algorithms**  Diploma in IT, ISF  Year 2 (2017/18) Semester 4 | **Week 4** |
| **1-2 Hours** |
| **Tutorial 4 – Stacks** | |

1. Suppose that s and t are empty stacks and a, b, c, and d are objects. What do the stacks contain after the following sequence of operations executes?

s.push(a);

s.push(b);

s.push(c);

t.push(d);

t.push(s.getTop());

s.pop();

t.push(s.getTop());

s.push(t.getTop());

t.pop();

t.pop();

|  |
| --- |
| S : a b b  T : d |

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

|  |
| --- |
| //stack.h  //array-based implementation  #ifndef STACK\_H  #define STACK\_H  #include <iostream>  #include <string>  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();  };  #endif |

Implement the following operations of the List ADT

1. ~Stack();

|  |
| --- |
|  |

1. bool pop(ItemType& item);

|  |
| --- |
|  |

1. void displayInOrder(); (without worry about changing the stack)

|  |
| --- |
|  |

*NB: The stack is empty after the above is executed.*

1. void displayInOrderOfInsertion();

|  |
| --- |
|  |

1. A stack is normally used in the translation of an infix expression to its postfix form. Show clearly, step-by-step, the contents of the stack and the contents of the postfix expression for each of the translation of infix to postfix expression below:

(a - b) / (c + d \* e) + f

|  |  |  |
| --- | --- | --- |
| **Next Character** | **Postfix Expression** | **Operator Stack** |
| ( |  | ( |
| a | a |  |
| - |  | ( - |
| b | a b | ( - |
| ) | A b | ( |
|  | A b - |  |
| / | A b - | / |
| ( | A b - | / ( |
| c | A b – c |  |
| + | A b – c | / ( + |
| d | A b – c d | /(+ |
| \* | A b – c d | /(+\* |
| e | A b – c d e | /(+\* |
| ) | A b – c d e \* | /(+ |
|  | A b – c d e \* + | /( |
|  | a b – c d e \* + | / |
| + | A b – c d e \* + / | + |
| f | A b – c d e \* + / f | + |
|  | a b – c d e \* + / f + |  |

1. An expression in postfix notation can be evaluated at run time by means of a stack. Show clearly the contents of the required stack when the following postfix expression is evaluated:

15 4 3 \* - 6 2 / +

|  |  |  |
| --- | --- | --- |
| **Next character** | **Stack** | **Evaluation** |
| 15 | 15 |  |
| 4 | 15 4 |  |
| 3 | 15 4 3 |  |
| \* | 15 | 4 \* 3 |
|  | 15 12 |  |
| - |  | 15-12 |
|  | 3 |  |
|  | 3 6 |  |
| 2 | 3 6 2 |  |
| / | 3 | 6/2 |
|  | 3 3 |  |
| + |  | 3+3 |
|  | 6 |  |