|  |  |
| --- | --- |
| **Data Structures & Algorithms**  Diploma in IT, ISF, FI  Year 2 (2018/19) 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();

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Stack s:** b b a  **Stack t:** d   |  |  | | --- | --- | | **Stack s** | **Stack t** | | b | d | | b |  | | a |  | |

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();  }; |

Implement the following operations of the List ADT

1. **~Stack();**

|  |
| --- |
| Stack::~Stack()  {  while(!isEmpty())  {  pop();  }  } |

1. **bool pop(ItemType& item);**

|  |
| --- |
| bool Stack::pop(ItemType &item)  {  if (!isEmpty())  {  Node\* delTemp;  delTemp = topNode;  topNode = topNode->next;  item = delTemp->item;  delete delTemp;  return true;  }  else  {  return false;  }  } |

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

|  |
| --- |
| void Stack::displayInOrder()  {  if (!isEmpty())  {  while (!isEmpty())  {  ItemType x;  pop(x);  cout << x << endl;  }  }  else  {  cout << "Nothing to display." << endl;  }  } |

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

1. **void displayInOrderOfInsertion();**

|  |
| --- |
| void Stack::displayInOrderOfInsertion()  {  if (!isEmpty())  {  Stack reverseStack;  Node\* tempNode;  tempNode = topNode;  while (tempNode != NULL)  {  reverseStack.push(tempNode->item);  tempNode = tempNode->next;  }  reverseStack.displayInOrder();  }  else  {  cout << "Nothing to display in order." << endl;  }  } |

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 | ( |
| - | a | ( - |
| b | a b | ( - |
| ) | ab- |  |
| / | ab- | / |
| ( | ab- | /( |
| c | ab-c | /( |
| + | ab-c | /(+ |
| d | ab-cd | /(+ |
| \* | ab-cd | /(+\* |
| e | ab-cde | /(+\* |
| ) | ab-cde\*+/ |  |
| + | ab-cde\*+/ | + |
| F | ab-cde\*+/f | + |
|  | ab-cde\*+/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 | 4  15 |  |
| 3 | 3  4  15 |  |
| \* | 15 | 4 \* 3 |
| - |  | 15 – 12 |
| 6 | 3  6 |  |
| 2 | 3  6  2 |  |
| / | 3 | 6 / 2 |
| + |  | 3 + 3 |