|  |  |
| --- | --- |
| **Data Structures & Algorithms**  Diploma in IT  Year 2 (2016/17) 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();

|  |
| --- |
|  |

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

|  |
| --- |
| //stack.h (Pointer-based implementation)  #pragma once  #include <iostream>  #include <string>  using namespace std;  typedef int ItemType;  class Stack  {  private:  struct Node  {  ItemType item;  Node \*next;  };  Node \*topNode;  public:  Stack(); //Default constructor    ~Stack(); //Destructor  //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 item from top of stack  ItemType getTop();  //display items in stack in order  void print();  //display items in stack in order of insertion  void printInOrderOfInsertion();  }; |

Removing all

Constructor:

Stack::Stack() // initialize attributes Stack only has topNode;

{

topNode = NULL;

}

Destructor

Stack::~Stack() // to release memory

{

Node \*temp = topNode;

While (temp-> next != NULL)

{

Node\* nextNode = temp->next; //1

Delete temp:

Temp = nextNode;

}

topNode = NULL;

}

Implement the following operations of the List ADT

1. bool pop();

|  |
| --- |
| Bool stack::pop()  {  if (isEmpty())  {  Node\* temp = topNode:  topNode = topNode->next; // topNode now points to 2nd item  temp->next = NULL;  delete temp;  return true;  }  return false;  }  bool Stack::pop(ItemType& item) // value popped stored in item  {  {  Node\* temp = topNode:  Item = temp->item; //store 1st item  topNode = topNode->next; // topNode now points to 2nd item  temp->next = NULL; // so old address not left in memory  delete temp; // no longer accessible in this program, memory contents still there.  return true;  }  return false;  }  bool Stack::pop(ItemType& item)  {  getTop(item);  pop();  } |

1. ItemType getTop();

|  |
| --- |
| Itemtype Stack::getTop()  {  ItemType item:  if(isEmpty())  item = topNode->item;  return item;  }  void Stack::getTop(ItemType& item)  {  } |

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

|  |
| --- |
|  |

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

1. void printInOrderOfInsertion();

|  |
| --- |
|  |

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 | ( - |
| ) | Ab- | ( |
|  | Ab- |  |
| / | Ab- | / |
| ( | Ab- | /( |
| C | Ab-c | /( |
| + | Ab-c | /(+ |
| D | Ab-cd | /(+ |
| \* | Ab-cd | /(+\* |
| E | Ab-cde | /(+\* |
| ) | Ab-cde\* | /(+ |
|  | Ab-cde\*+ | /( |
|  | 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 |
|  | 12  15 |  |
| - |  | 15-12 |
|  | 3 |  |
| 6 | 6  3 |  |
| 2 | 2  6  3 |  |
| / | 3 | 6/2 |
|  | 3  3 |  |
| + |  | 3+3 |
|  | 6 |  |