|  |  |
| --- | --- |
| **Data Structures & Algorithms**  Diploma in CSF, IT  Year 2/3 (2020/21) Semester 4/6 | **Week 4** |
| **1 Hour** |
| **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>  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();  }  }  Stack::Stack(){  While(topNode != NULL){  Pop();  }  }  Stack::Stack(){  Node \*tmp = topNode;  While (topNode != NULL){    topNode = topNode->next;  Delete tmp;  Tmp = NULL;  }  } |

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

|  |
| --- |
| Bool Stack::pop(){  Bool success = !isEmpty();  If(success){  Node \*temp = topNode;  topNode = topNode->next;  //return deleted node to system  item = temp->item;  Temp->next = NULL  Delete temp;  Temp = NULL;  }  }  Bool Stack::pop(ItemType item){  Bool success = !isEmpty();  If(success){  Node \*temp = topNode;  topNode = topNode->next;  //return deleted node to system  // if func include item  item = &(temp->item);  Temp->next = NULL  Delete temp;  Temp = NULL;  }  } |

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

|  |
| --- |
| Void Stack::displayInOrder(){  ItemType item;  If(!isEmpty()){  While(!isEmpty()){  getTop(item);  cout << item << endl;  pop();  //pop(item);  //cout << item << endl;  }  }  }  Void Stack::displayInOrder(){  ItemType item;  If(topNode != NULL){  Node \*temp = topNode;  While(temp != NULL){  cout << temp->item << endl;  temp = temp->next;  }  }  }  Void Stack::displayInOrder(){  ItemType item;  Stack s;  If(!isEmpty()){  //pop from temp stack and restore contents of original  //stack  While(!isEmpty()){  s.getTop(item)  cout << item << endl;  s.pop();  push(item);  //s.pop(item);  //push(item);  }  }  } |

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

1. **void displayInOrderOfInsertion();**

|  |
| --- |
| Void Stack::displayInOrderOfInsertion(){  ItemType item;  Stack tempStack;  If (!isEmpty()){  While(!isEmpty()){  getTop(item);  tempStack.push(item);  pop();  }  While (!tempStack.isEmpty()){  tempStack.getTop(item);  }  }  } |

1. Using a stack, implement the **reverse()** function that takes a string as an argument and returns the string with its characters reversed.

For example: **reverse("abcde")** returns **"edcba"**.

The function prototype is:

**string reverse(string);**

String Stack::reverse(string input){

String reverse = string(input.rbegin() , input.rend());

Return reverse;

}