|  |  |
| --- | --- |
| **++Data Structures & Algorithms**  Diploma in IT, CSF  Year 2 (2024/25) Semester 3 | **Week 4** |
| **1 Hour** |
| **Tutorial 4 – Stacks** | |

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

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?
2. s.push(a);
3. s.push(b);
4. s.push(c);
5. t.push(d);
6. t.push(s.getTop());
7. s.pop();
8. t.push(s.getTop());
9. s.push(t.getTop());
10. t.pop();
11. t.pop();

|  |
| --- |
| 1. Stack s [a] stack t [] 2. Stack s [a, b] stack t [] 3. Stack s [a, b, c] stack t [] 4. Stack t [d] stack s [a, b, c] 5. Stack t [d, c] stack s [a, b, c] 6. Stack s [a, b] stack t [d, c] 7. Stack t [d, c, b] stack s [a, b] 8. Stack s [a, b, b] stack t [d, c, b] 9. Stack t [d, c], stack s [a, b, b] 10. Stack t [d], stack s [a, b, b] |

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 Stack ADT

1. **~Stack();**

|  |
| --- |
| // Destructor  Stack::~Stack()  {      while (!isEmpty())      {          pop();      }  }  bool Stack::pop()  {      if (isEmpty())      {          return false; // Stack is empty      }      else      {          Node \*temp = topNode;    // Save the top node          topNode = topNode->next; // Update topNode to the next node          delete temp;             // Delete the top node          return true;      }  } |

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

|  |
| --- |
| bool Stack::pop(ItemType &item)  {      if (isEmpty())      {          return false; // Stack is empty      }      else      {          item = topNode->item; // Retrieve the item          return pop();         // Pop the top node      }  } |

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

|  |
| --- |
| void Stack::displayInOrder()  {      if (isEmpty())      {          cout << "Stack is empty." << endl;      }      else      {          Node \*temp = topNode;          while (temp != nullptr)          {              cout << temp->item << " ";              temp = temp->next;          }      }      cout << endl;  }  --- void Stack::displayInOrder()      {          if (isEmpty())          {              cout << "Stack is empty." << endl;              return;          }          Stack tempStack; // Temporary stack to hold elements          // Pop from the original stack and push into the temporary stack          while (!isEmpty())          {              int value = pop();              tempStack.push(value);          }          // Display the elements in the temporary stack and push them back into the original stack          while (!tempStack.isEmpty())          {              int value = tempStack.pop();              cout << value << " "; // Display the value              push(value);          // Push it back into the original stack          }          cout << endl;      } |
|  |

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

1. **void displayInOrderOfInsertion();**

|  |
| --- |
| void Stack::displayInOrderOfInsertion() {      if (isEmpty())      {          cout << "Stack is empty." << endl;      }      else      {          Stack tempStack;          Node\* current = topNode;          // Push all items to a temporary stack          while (current != nullptr)          {              tempStack.push(current->item);              current = current->next;          }          // Display items in the temporary stack          tempStack.displayInOrder();      }  } |

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);**

include <stack>

#include <string>

using namespace std;

string reverse(const string& str) {

    stack<char> charStack;

    string reversedString;

    // Push each character of 'str' onto the stack

    for (char ch : str) {

        charStack.push(ch);

    }

    // Pop from the stack and append to 'reversedString'

    while (!charStack.empty()) {

        reversedString += charStack.top();  // Get the top character

        charStack.pop();                    // Remove the top character from the stack

    }

    return reversedString;

}

int main() {

    // Test the reverse function

    string input = "abcde";

    string output = reverse(input);

    cout << "Reversed string: " << output << endl;  // Expected output: "edcba"

    return 0;

}