**CSC 2201 – Computer Science II**

**Shivang Trivedi**

**Lab #06**

**Due: Wednesday 27th Feb, 11:00 am**

**60 Points**

1. Save a copy of this document with your name and the assignment number somewhere in the file name. For example, the file name *“CSC2201\_Doe\_Jane\_Lab6.docx”*
2. Copy-and-paste your answers (e.g., C++ source code) into the document.
3. Copy-and-paste the program output window.
4. Submit the following files separately (do not compress the files) to the Canvas in one submission:

1) This document as a word document (i.e., with the extension ***.doc*** or ***.docx***).

2) All C++ source code solution file(s) (only the ***.cpp*** and ***.h*** files) to the Canvas item associated with this assignment/lab solution.

Questions:

- implement the Stack ADT using the linked list approach:

- implement constructor, copy constructor, assignment operator, destructor; (25 pts)

- push, pop; (25 pts)

- clear , isFull, isEmpty; (10 pts)

#include <stdexcept>

#include <iostream>

using namespace std;

#include "StackLinked.h"

//Name: Shivang Trivedi

//Date: Feb 26, 2019

//Description: This program creates a stack using linked list

template<typename DataType>

inline StackLinked<DataType>::StackLinked(int maxNumber)

{

//initialize top

top = NULL;

}

template<typename DataType>

StackLinked<DataType>::StackLinked(const StackLinked & other)

{

//create a new stacklinked type and set it to others

StackNode \*otherPtr = other.top;

//while the other is not zero, insert the new item

while (otherPtr != 0) {

push(other->dataItem);

otherPtr = otherPtr->next;

}

}

template<typename DataType>

StackLinked<typename DataType> & StackLinked<DataType>::operator=(const StackLinked & other)

{

//if they are not the same then copy it

if (this != &other) return \*this;

top = new StackNode(other.top->dataItem, 0);

StackNode \*otherTemp = other.top->next;

StackNode \*thisTemp = 0;

StackNode \*thisPrevious = top;

while (otherTemp != 0)

{

thisTemp = new StackNode(otherTemp->dataItem, 0);

thisPrevious->next = 0;

thisPrevious = top->next;

otherTemp = top;

}

}

template<typename DataType>

StackLinked<DataType>::~StackLinked()

{

StackNode \*x;

//delete each node until it equals to 0

while (top != NULL) {

x = top;

top = top->next;

delete x;

}

}

template<typename DataType>

void StackLinked<DataType>::push(const DataType & newDataItem) throw(logic\_error)

{

if (isFull())

throw logic\_error("The list is full\n");

else if (top == NULL)

top = new StackNode(newDataItem, 0);

else {

//insert a new node

StackNode \*temp;

temp = new StackNode(newDataItem, top);

temp->next = top;

top = temp;

}

}

template<typename DataType>

DataType StackLinked<DataType>::pop() throw(logic\_error)

{

StackNode \*temp;

if (top == NULL) {

throw logic\_error("The list is empty");

}

else {

temp = top;

top = top->next;

return temp->dataItem;

}

}

template<typename DataType>

void StackLinked<DataType>::clear()

{

// create a temp and delete all node

StackNode \*temp;

while (top != NULL) {

temp = top;

top = top->next;

delete temp;

}

}

template<typename DataType>

bool StackLinked<DataType>::isEmpty() const

{

return(top == NULL);

}

template<typename DataType>

bool StackLinked<DataType>::isFull() const

{

return false;

}

template<typename DataType>

void StackLinked<DataType>::showStructure() const

{

// Linked list implementation. Outputs the data elements in a stack.

// If the stack is empty, outputs "Empty stack". This operation is

// intended for testing and debugging purposes only.

{

if (isEmpty())

{

cout << "Empty stack" << endl;

}

else

{

cout << "Top\t";

for (StackNode\* temp = top; temp != 0; temp = temp->next) {

if (temp == top) {

cout << "[" << temp->dataItem << "]\t";

}

else {

cout << temp->dataItem << "\t";

}

}

cout << "Bottom" << endl;

}

}

}

