Jessica Sullivan

ID: 1282151

Class: COMSC-210-5067

Professor: Pentcheva

Source Code:

CS210\_Assignment5A\_Stack.cpp:

*/\**

*Programmer: Jessica Sullivan*

*Programmer's ID: 1282151*

*Class: COMSC-210-5067*

*\*/*

*// main.cpp*

*// CS210\_Assignment5\_StackQueue*

*//*

*// Created by Jessie Sully on 3/7/20.*

*// Copyright © 2020 Jessie Sully. All rights reserved.*

*//*

#include <iostream>

#include "Node.h"

#include "Stack.h"

**bool** testMemoryLeak();

**template** <**typename** T>

**void** display(**const** Stack<T>& stack);

**void** runProgram();

**int** main(**int** argc, **const** **char** \* argv[]) {

*// programmer's identification*

cout << "Programmer: Jessica Sullivan" << endl;

cout << "Programmer's ID: 1282151" << endl;

cout << "File: " << \_\_FILE\_\_ << endl;

runProgram();

testMemoryLeak();

**return** 0;

}

*// runs program*

**void** runProgram() {

**int** choice, data;

Stack<**int**> stack;

**while**(1) {

*/\* Menu \*/*

cout <<"------------------------------------\n";

cout <<" STACK IMPLEMENTATION PROGRAM \n";

cout <<"------------------------------------\n";

cout <<"1. Push\n";

cout <<"2. Pop\n";

cout <<"3. Size\n";

cout <<"4. Print Stack\n";

cout <<"5. Exit\n";

cout <<"------------------------------------\n";

cout <<"Enter your choice: ";

cin >>choice;

**switch**(choice) {

**case** 1:

cout <<"Enter data to push into stack: ";

cin >> data;

stack.push(data);

**break**;

**case** 2:

**if** (stack.isEmpty()) {

cerr << "Error, stack is empty." << endl;

}

**else** {

data = stack.pop();

cout <<"Data => " << data << endl;

}

**break**;

**case** 3:

cout << "Stack size: " << stack.getSize() << endl;

**break**;

**case** 4:

display(stack);

**break**;

**case** 5:

cout <<"Exiting from app.\n";

exit(0);

**break**;

**default**:

cout <<"Invalid choice, please try again.\n";

}

cout <<"\n\n";

}

}

*// returns true if memory leak*

**bool** testMemoryLeak() {

**if** (sNumNodeObjects != 0) {

cerr << "Num leaked nodes: " << sNumNodeObjects << endl;

**return** **true**;

}

**return** **false**;

}

*// Function to display elements in stack*

**template** <**typename** T>

**void** display(**const** Stack<T>& stack) {

**if** (!stack.isEmpty()) {

stack.display(cout);

cout << endl;

}

**else**

cout << "stack is empty" << endl << endl;

}

Node.h:

*/\**

*Programmer: Jessica Sullivan*

*Programmer's ID: 1282151*

*Class: COMSC-210-5067*

*\*/*

*// NODE.h*

*// CS260\_Assignment4\_LinkedLists*

*//*

*// Created by Jessie Sully on 2/25/20.*

*// Copyright © 2020 Jessie Sully. All rights reserved.*

*//*

#ifndef NODE\_H

#define NODE\_H

**using** **namespace** std;

*// static to keep track of instances to check memory leak. Keeps track of all variants.*

**static** **int** sNumNodeObjects = 0;

**template** <**typename** T>

**struct** Node {

**public**:

Node() { sNumNodeObjects++; }; *// update memory leak tracking number*

Node(T object) : mObject(object) { sNumNodeObjects++; } *// update memory leak tracking number*

~Node();

**template** <**typename** U>

**friend** ostream& **operator**<<(ostream& ostr, **const** Node<U>\* node);

T mObject;

Node\* nextNode = **nullptr**;

Node\* prevNode = **nullptr**;

};

**template** <**typename** T>

Node<T>::~Node() {

*// unlink node*

**if** (nextNode != **nullptr**) {

nextNode->prevNode = prevNode;

}

**if** (prevNode != **nullptr**) {

prevNode->nextNode = nextNode;

}

prevNode = **nullptr**;

nextNode = **nullptr**;

*// update memory leak tracking number*

sNumNodeObjects--;

}

**template**<**typename** U>

ostream& **operator**<<(ostream& ostr, **const** Node<U>\* node) {

ostr << node->mObject;

**return** ostr;

}

#endif */\* NODE\_H \*/*

Stack.h:

*/\**

*Programmer: Jessica Sullivan*

*Programmer's ID: 1282151*

*Class: COMSC-210-5067*

*\*/*

*// Stack.h*

*// CS210\_Assignment5\_StackQueue*

*//*

*// Created by Jessie Sully on 3/7/20.*

*// Copyright © 2020 Jessie Sully. All rights reserved.*

*//*

#ifndef Stack\_h

#define Stack\_h

#include "Node.h"

**template** <**typename** T>

**class** Stack {

**public**:

Stack(){}

~Stack();

**void** push(T element);

T pop();

**bool** isEmpty() **const** { **return** mHead == **nullptr**; }

**void** display(ostream& output) **const**;

**int** getSize() **const** { **return** mSize; }

**private**:

Node<T>\* mHead = **nullptr**;

**int** mSize = 0;

};

**template** <**typename** T>

Stack<T>::~Stack() {

**while** (mHead != **nullptr**) {

mHead = mHead->nextNode;

**delete** mHead->prevNode;

}

}

**template** <**typename** T>

**void** Stack<T>::push(T element) {

Node<T>\* newNode = **new** Node<T>(element);

**if** (mHead == **nullptr**) {

mHead = newNode;

}

**else** {

newNode->nextNode = mHead;

newNode->prevNode = **nullptr**;

mHead->prevNode = newNode;

mHead = newNode;

}

mSize++;

}

*//Function to pop element from top of stack.*

**template** <**typename** T>

T Stack<T>::pop() {

**if** (isEmpty()) {

cerr << "Error, stack is empty." << endl;

T element = T();

**return** element;

}

**else** {

T element = mHead->mObject;

**if** (mSize == 1) {

**delete** mHead;

mHead = **nullptr**;

}

**else** {

mHead = mHead->nextNode;

**delete** mHead->prevNode;

mHead->prevNode = **nullptr**;

}

mSize--;

**return** element;

}

}

**template**<**typename** T>

**void** Stack<T>::display(ostream &output) **const** {

**for**(**const** Node<T>\* cursor = mHead; cursor != **nullptr**;

cursor = cursor->nextNode) {

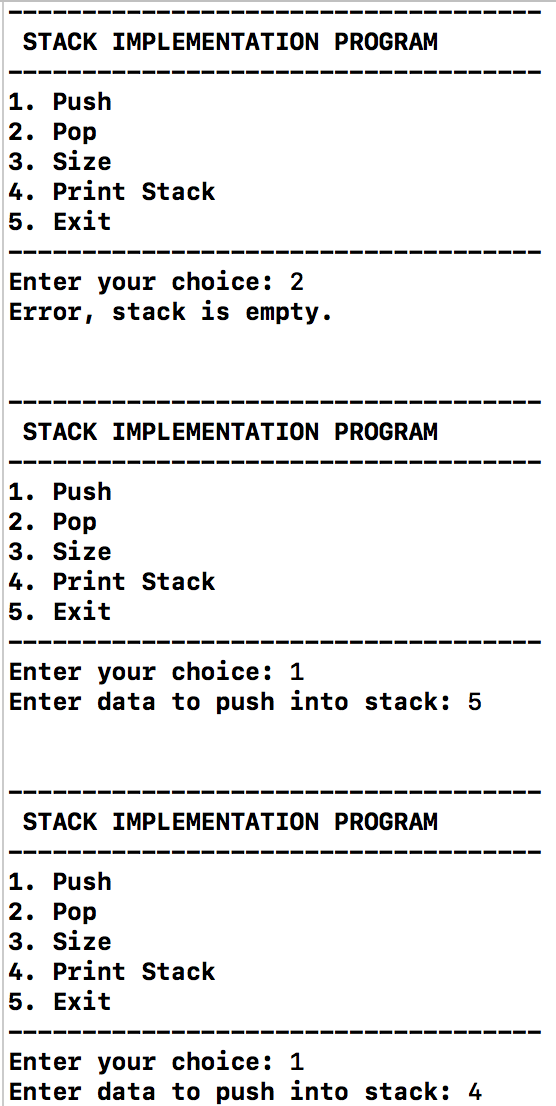
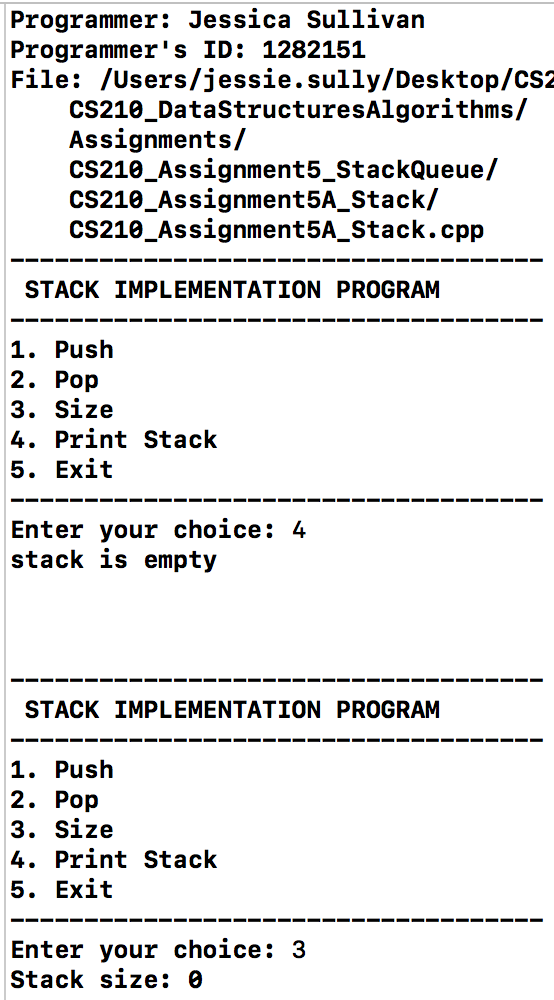
output << cursor << endl;

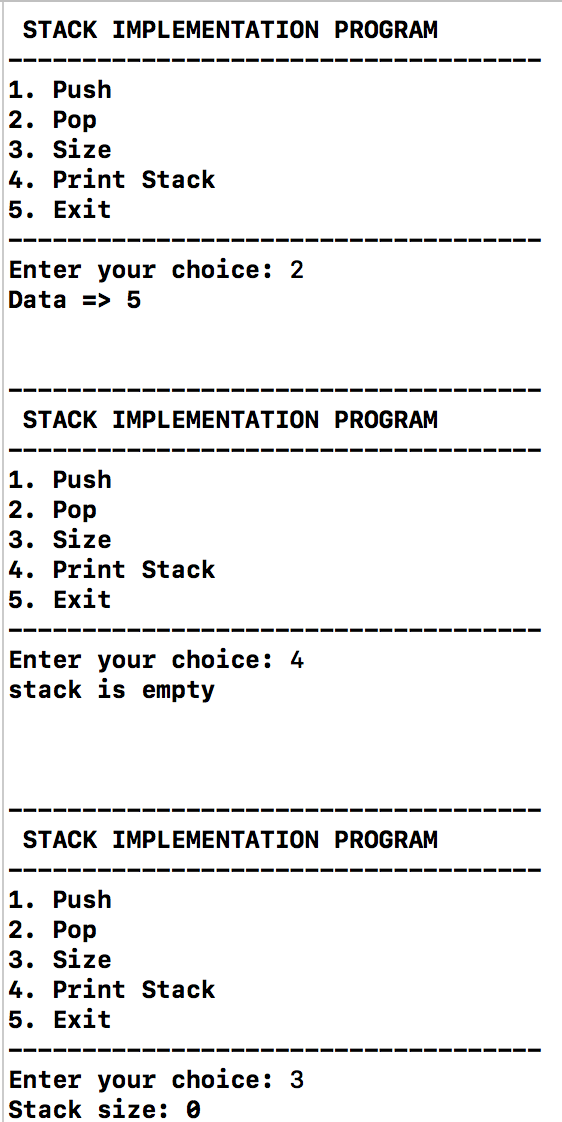
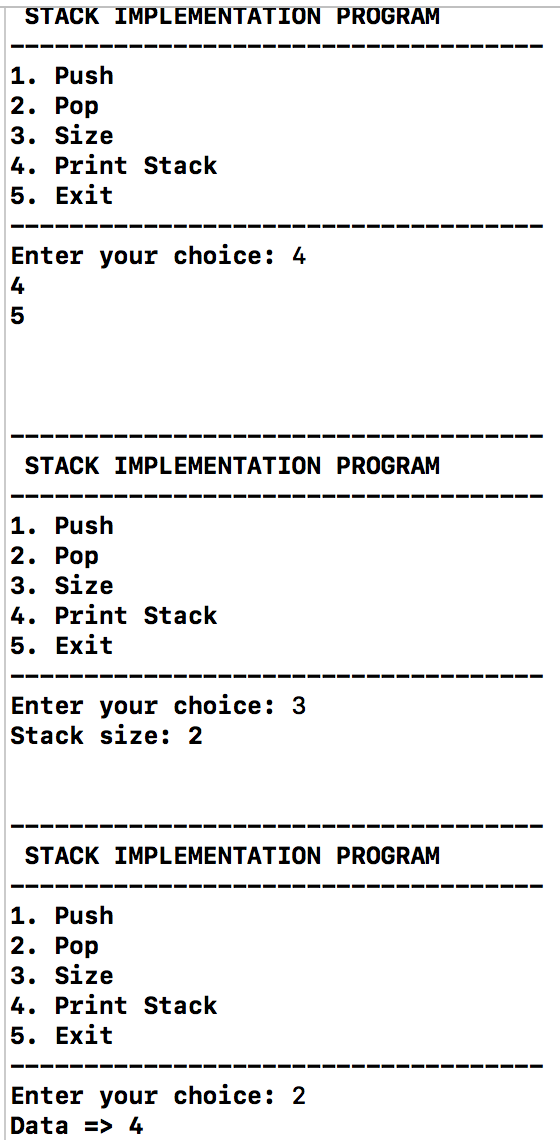
}

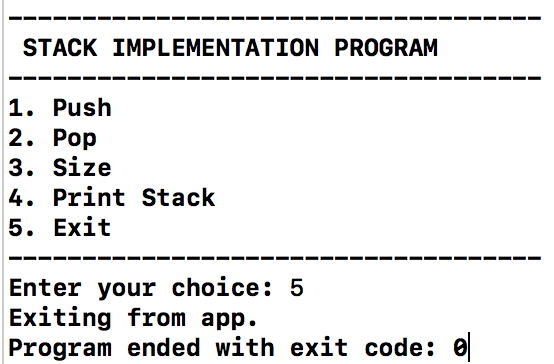
}

#endif */\* Stack\_h \*/*

Output:







Source Code:

CS210\_Assignment5B\_Queue.cpp:

*/\**

*Programmer: Jessica Sullivan*

*Programmer's ID: 1282151*

*Class: COMSC-210-5067*

*\*/*

*// main.cpp*

*// CS210\_Assignment5\_QueueQueue*

*//*

*// Created by Jessie Sully on 3/7/20.*

*// Copyright © 2020 Jessie Sully. All rights reserved.*

*//*

#include <iostream>

#include "Node.h"

#include "Queue.h"

**bool** testMemoryLeak();

**template** <**typename** T>

**void** display(**const** Queue<T>& queue);

**void** runProgram();

string prepMenu();

**int** main(**int** argc, **const** **char** \* argv[]) {

*// programmer's identification*

cout << "Programmer: Jessica Sullivan" << endl;

cout << "Programmer's ID: 1282151" << endl;

cout << "File: " << \_\_FILE\_\_ << endl;

runProgram();

testMemoryLeak();

**return** 0;

}

*// returns true if memory leak*

**bool** testMemoryLeak() {

**if** (sNumNodeObjects != 0) {

cerr << "Num leaked nodes: " << sNumNodeObjects << endl;

**return** **true**;

}

**return** **false**;

}

*// Function to display elements in queue*

**template** <**typename** T>

**void** display(**const** Queue<T>& queue) {

**if** (!queue.isEmpty()) {

queue.display(cout);

cout << endl;

}

**else** {

cout << "queue is empty" << endl << endl;

}

}

**void** runProgram() {

**int** option, data;

Queue<**int**> queue;

string menu = prepMenu();

cout << menu << endl;

cin >> option;

**while** (option !=7) {

**switch** (option) {

**case** 1:

cout << "\nEnter data to enqueue (-99 to stop): ";

cin >> data;

**while** ( data != -99) {

queue.enqueue(data);

cout << "\nEnter data to enqueue (-99 to stop): ";

cin >> data;

}

**break**;

**case** 2:

**if** (queue.isEmpty()) {

cout << "Queue is empty."<< endl;

}

**else** {

data = queue.dequeue();

cout << "Data => " << data << endl;

}

**break**;

**case** 3:

**if** (queue.isEmpty()) {

cout << "Queue is empty."<< endl;

}

**else** {

cout << "Queue size => "<< queue.getSize() << endl;

}

**break**;

**case** 4:

**if** (queue.isEmpty()) {

cout << "Queue is empty."<< endl;

}

**else** {

data = queue.getRear();

cout << "Rear => " << data << endl;

}

**break**;

**case** 5:

**if** (queue.isEmpty()) {

cout << "Queue is empty."<< endl;

}

**else** {

data = queue.getFront();

cout <<"Front => " << data << endl;

}

**break**;

**case** 6:

display(queue);

**break**;

**default**:

cout <<"Invalid choice, please input number between (0-5).\n";

**break**;

}

cout <<"\n\n";

cout << menu<< endl;

cin >> option;

}

}

string prepMenu() {

string menu = "";

menu+= " \n-------------------------------------------------------------------\n";

menu+= "1.Enqueue 2.Dequeue 3.Size 4.Get Rear 5.Get Front 6.Display 7.Exit\n";

menu+= "----------------------------------------------------------------------\n";

menu+= "Select an option: ";

**return** menu;

}

Node.h:

*/\**

*Programmer: Jessica Sullivan*

*Programmer's ID: 1282151*

*Class: COMSC-210-5067*

*\*/*

*// NODE.h*

*// CS260\_Assignment4\_LinkedLists*

*//*

*// Created by Jessie Sully on 2/25/20.*

*// Copyright © 2020 Jessie Sully. All rights reserved.*

*//*

#ifndef NODE\_H

#define NODE\_H

**using** **namespace** std;

*// static - Keeps track of number of instances to check for memory leak, keeps track of all variants.*

**static** **int** sNumNodeObjects = 0;

**template** <**typename** T>

**struct** Node {

**public**:

Node() { sNumNodeObjects++; }; *// update memory leak tracking number*

Node(T object) : mObject(object) { sNumNodeObjects++; } *// update memory leak tracking number*

~Node();

**template** <**typename** U>

**friend** ostream& **operator**<<(ostream& ostr, **const** Node<U>\* node);

T mObject;

Node\* nextNode = **nullptr**;

Node\* prevNode = **nullptr**;

};

**template** <**typename** T>

Node<T>::~Node() {

*// unlink node*

**if** (nextNode != **nullptr**) {

nextNode->prevNode = prevNode;

}

**if** (prevNode != **nullptr**) {

prevNode->nextNode = nextNode;

}

prevNode = **nullptr**;

nextNode = **nullptr**;

*// update memory leak tracking number*

sNumNodeObjects--;

}

**template**<**typename** U>

ostream& **operator**<<(ostream& ostr, **const** Node<U>\* node) {

ostr << node->mObject;

**return** ostr;

}

#endif */\* NODE\_H \*/*

Queue.h:

*/\**

*Programmer: Jessica Sullivan*

*Programmer's ID: 1282151*

*Class: COMSC-210-5067*

*\*/*

*// Queue.h*

*// CS210\_Assignment5\_Queue*

*//*

*// Created by Jessie Sully on 3/7/20.*

*// Copyright © 2020 Jessie Sully. All rights reserved.*

*//*

#ifndef Queue\_h

#define Queue\_h

#include "Node.h"

**template** <**typename** T>

**class** Queue {

**public**:

Queue(){}

~Queue();

**void** enqueue(T element);

T dequeue();

**bool** isEmpty() **const** { **return** mHead == **nullptr**; }

**void** display(ostream& output) **const**;

**int** getSize() **const** { **return** mSize; }

**const** T& getRear() **const** { **return** mTail->mObject; }

**const** T& getFront() **const** { **return** mHead->mObject; }

**private**:

Node<T>\* mHead = **nullptr**;

Node<T>\* mTail = **nullptr**;

**int** mSize = 0;

};

**template** <**typename** T>

Queue<T>::~Queue() {

**while** (mHead != **nullptr**) {

mHead = mHead->nextNode;

**delete** mHead->prevNode;

}

}

**template** <**typename** T>

**void** Queue<T>::enqueue(T element) {

Node<T>\* newNode = **new** Node<T>(element);

**if** (mHead == **nullptr**) {

mHead = newNode;

mTail = newNode;

}

**else** {

newNode->nextNode = **nullptr**;

newNode->prevNode = mTail;

mTail->nextNode = newNode;

mTail = newNode;

}

mSize++;

}

*//Function to dequeue element from top of stack.*

**template** <**typename** T>

T Queue<T>::dequeue() {

**if** (isEmpty()) {

T element = T();

**return** element;

}

**else** {

T element = mHead->mObject;

**if** (mSize == 1) {

**delete** mHead;

mHead = **nullptr**;

mTail = **nullptr**;

}

**else** {

mHead = mHead->nextNode;

**delete** mHead->prevNode;

}

mSize--;

**return** element;

}

}

**template**<**typename** T>

**void** Queue<T>::display(ostream &output) **const** {

**for**(Node<T>\* cursor = mHead; cursor != **nullptr**; cursor = cursor->nextNode) {

output << cursor << endl;

}

}

#endif */\* Queue\_h \*/*

Output:

