Jessica Sullivan

ID: 1282151

Class: COMSC-210-5067

Professor: Pentcheva

Source Code:

CS210\_Assignment6\_Recursion.cpp:

*/\**

*Programmer: Jessica Sullivan*

*Programmer's ID: 1282151*

*Class: COMSC-210-5067*

*\*/*

*// main.cpp*

*// CS210\_Assignment6\_Recursion*

*//*

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

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

*//*

#include <iostream>

#define VERBOSE

#include "Node.h"

#include "Stack.h"

#include "HannoiTower.h"

**void** runProgram();

**bool** testMemoryLeak();

**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** numDisks = 0;

cout << "TOWERS of Hanoi\nEnter a number of disks to play. I'll give necessary moves." << endl;

cin >> numDisks;

HanoiTower tower(numDisks);

tower.printStacks();

tower.moveStack(tower.mFirst, tower.mSecond, tower.mThird, numDisks);

}

*// returns true if memory leak*

**bool** testMemoryLeak() {

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

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

**return** **true**;

}

**return** **false**;

}

HannoiTower.h:

*/\**

*Programmer: Jessica Sullivan*

*Programmer's ID: 1282151*

*Class: COMSC-210-5067*

*\*/*

*// HannoiTower.h*

*// CS210\_Assignment6\_Recursion*

*//*

*// Created by Jessie Sully on 3/19/20.*

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

*//*

#ifndef HannoiTower\_h

#define HannoiTower\_h

**struct** HanoiTower {

HanoiTower(**int** numDisks);

Stack<**unsigned** **int**> mFirst;

Stack<**unsigned** **int**> mSecond;

Stack<**unsigned** **int**> mThird;

**void** printStacks();

**void** moveStack(Stack<**unsigned** **int**> &source, Stack<**unsigned** **int**> &destination, Stack<**unsigned** **int**> &temp, **unsigned** **int** numDisks);

};

HanoiTower::HanoiTower(**int** numDisks) : mFirst("A"), mSecond("B"), mThird("C") {

**for** (**int** i = numDisks; i > 0; i--) {

mFirst.push(i);

}

}

**void** HanoiTower::printStacks() {

#ifdef VERBOSE

mFirst.display(cout);

mSecond.display(cout);

mThird.display(cout);

#endif *// VERBOSE*

}

**void** HanoiTower::moveStack(Stack<**unsigned** **int**> &source, Stack<**unsigned** **int**> &destination, Stack<**unsigned** **int**> &temp, **unsigned** **int** numDisks) {

**if** (numDisks == 0) {

**return**;

}

**else** {

**for** (**int** k = 1; k <= numDisks; k++) {

moveStack(destination, temp, source, k - 1);

destination.push(source.pop());

cout << "Move Disk " << k << " from post " << source.getName() << " to post " << destination.getName() << endl;

printStacks();

moveStack(temp, destination, source, k - 1);

}

}

}

#endif */\* HannoiTower\_h \*/*

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 <string>

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

#include "Node.h"

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

**class** Stack {

**public**:

Stack(){}

Stack(**const** string &name) : mName(name){}

~Stack();

**void** push(T element);

T pop();

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

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

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

**const** string& getName() { **return** mName; }

**void** setName(**const** string& name) { mName = name; }

T getHead() { **return** mHead != **nullptr** ? mHead->mObject : 0; }

**private**:

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

**int** mSize = 0;

string mName = "";

};

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

Stack<T>::~Stack() {

Node<T>\* next;

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

next = mHead->nextNode;

**delete** mHead;

mHead = next;

}

}

**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** {

output << mName << " : ";

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

cursor = cursor->nextNode) {

output << cursor << ' ';

}

output << endl;

}

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

Output Run 1:

**Programmer: Jessica Sullivan**

**Programmer's ID: 1282151**

**File: /Users/jessie.sully/Desktop/CS210/CS210\_DataStructuresAlgorithms/Assignments/CS210\_Assignment6\_Recursion/CS210\_Assignment6\_Recursion/CS210\_Assignment6\_Recursion.cpp**

**TOWERS of Hanoi**

**Enter a number of disks to play. I'll give necessary moves.**

3

**A : 1 2 3**

**B :**

**C :**

**Move Disk 1 from post A to post B**

**A : 2 3**

**B : 1**

**C :**

**Move Disk 1 from post B to post C**

**A : 2 3**

**B :**

**C : 1**

**Move Disk 2 from post A to post B**

**A : 3**

**B : 2**

**C : 1**

**Move Disk 1 from post C to post B**

**A : 3**

**B : 1 2**

**C :**

**Move Disk 1 from post B to post C**

**A : 3**

**B : 2**

**C : 1**

**Move Disk 1 from post C to post A**

**A : 1 3**

**B : 2**

**C :**

**Move Disk 2 from post B to post C**

**A : 1 3**

**B :**

**C : 2**

**Move Disk 1 from post A to post C**

**A : 3**

**B :**

**C : 1 2**

**Move Disk 3 from post A to post B**

**A :**

**B : 3**

**C : 1 2**

**Move Disk 1 from post C to post B**

**A :**

**B : 1 3**

**C : 2**

**Move Disk 1 from post B to post A**

**A : 1**

**B : 3**

**C : 2**

**Move Disk 2 from post C to post B**

**A : 1**

**B : 2 3**

**C :**

**Move Disk 1 from post A to post B**

**A :**

**B : 1 2 3**

**C :**

**Program ended with exit code: 0**

Output Run 2:

**Programmer: Jessica Sullivan**

**Programmer's ID: 1282151**

**File: /Users/jessie.sully/Desktop/CS210/CS210\_DataStructuresAlgorithms/Assignments/CS210\_Assignment6\_Recursion/CS210\_Assignment6\_Recursion/CS210\_Assignment6\_Recursion.cpp**

**TOWERS of Hanoi**

**Enter a number of disks to play. I'll give necessary moves.**

4

**A : 1 2 3 4**

**B :**

**C :**

**Move Disk 1 from post A to post B**

**A : 2 3 4**

**B : 1**

**C :**

**Move Disk 1 from post B to post C**

**A : 2 3 4**

**B :**

**C : 1**

**Move Disk 2 from post A to post B**

**A : 3 4**

**B : 2**

**C : 1**

**Move Disk 1 from post C to post B**

**A : 3 4**

**B : 1 2**

**C :**

**Move Disk 1 from post B to post C**

**A : 3 4**

**B : 2**

**C : 1**

**Move Disk 1 from post C to post A**

**A : 1 3 4**

**B : 2**

**C :**

**Move Disk 2 from post B to post C**

**A : 1 3 4**

**B :**

**C : 2**

**Move Disk 1 from post A to post C**

**A : 3 4**

**B :**

**C : 1 2**

**Move Disk 3 from post A to post B**

**A : 4**

**B : 3**

**C : 1 2**

**Move Disk 1 from post C to post B**

**A : 4**

**B : 1 3**

**C : 2**

**Move Disk 1 from post B to post A**

**A : 1 4**

**B : 3**

**C : 2**

**Move Disk 2 from post C to post B**

**A : 1 4**

**B : 2 3**

**C :**

**Move Disk 1 from post A to post B**

**A : 4**

**B : 1 2 3**

**C :**

**Move Disk 1 from post B to post C**

**A : 4**

**B : 2 3**

**C : 1**

**Move Disk 1 from post C to post A**

**A : 1 4**

**B : 2 3**

**C :**

**Move Disk 2 from post B to post C**

**A : 1 4**

**B : 3**

**C : 2**

**Move Disk 1 from post A to post C**

**A : 4**

**B : 3**

**C : 1 2**

**Move Disk 1 from post C to post A**

**A : 1 4**

**B : 3**

**C : 2**

**Move Disk 1 from post A to post B**

**A : 4**

**B : 1 3**

**C : 2**

**Move Disk 2 from post C to post A**

**A : 2 4**

**B : 1 3**

**C :**

**Move Disk 1 from post B to post A**

**A : 1 2 4**

**B : 3**

**C :**

**Move Disk 3 from post B to post C**

**A : 1 2 4**

**B :**

**C : 3**

**Move Disk 1 from post A to post C**

**A : 2 4**

**B :**

**C : 1 3**

**Move Disk 1 from post C to post B**

**A : 2 4**

**B : 1**

**C : 3**

**Move Disk 2 from post A to post C**

**A : 4**

**B : 1**

**C : 2 3**

**Move Disk 1 from post B to post C**

**A : 4**

**B :**

**C : 1 2 3**

**Move Disk 4 from post A to post B**

**A :**

**B : 4**

**C : 1 2 3**

**Move Disk 1 from post C to post B**

**A :**

**B : 1 4**

**C : 2 3**

**Move Disk 1 from post B to post A**

**A : 1**

**B : 4**

**C : 2 3**

**Move Disk 2 from post C to post B**

**A : 1**

**B : 2 4**

**C : 3**

**Move Disk 1 from post A to post B**

**A :**

**B : 1 2 4**

**C : 3**

**Move Disk 1 from post B to post A**

**A : 1**

**B : 2 4**

**C : 3**

**Move Disk 1 from post A to post C**

**A :**

**B : 2 4**

**C : 1 3**

**Move Disk 2 from post B to post A**

**A : 2**

**B : 4**

**C : 1 3**

**Move Disk 1 from post C to post A**

**A : 1 2**

**B : 4**

**C : 3**

**Move Disk 3 from post C to post B**

**A : 1 2**

**B : 3 4**

**C :**

**Move Disk 1 from post A to post B**

**A : 2**

**B : 1 3 4**

**C :**

**Move Disk 1 from post B to post C**

**A : 2**

**B : 3 4**

**C : 1**

**Move Disk 2 from post A to post B**

**A :**

**B : 2 3 4**

**C : 1**

**Move Disk 1 from post C to post B**

**A :**

**B : 1 2 3 4**

**C :**

**Program ended with exit code: 0**