Jessica Sullivan

ID: 1282151

Class: COMSC-210-5067

Professor: Pentcheva

Source Code:

CS210\_Assignment9\_Heaps.cpp:

*/\**

*Programmer: Jessica Sullivan*

*Programmer's ID: 1282151*

*Class: COMSC-210-5067*

*\*/*

*// main.cpp*

*// CS210\_Assignment9\_Heaps*

*//*

*// Created by Jessie Sully on 4/17/20.*

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

*//*

#include <assert.h>

#include <iostream>

#include <fstream>

#include <sstream>

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

*//#define RUN\_TESTS*

*//#define RUN\_INPUT\_TEST*

#include "PriorityQueue.h"

#ifdef RUN\_INPUT\_TEST

**static** **const** string TEST\_INPUT = "input.txt";

#if defined(WIN32) || defined(\_WIN32)

#define PATH\_SEPARATOR "\\"

#else

#define PATH\_SEPARATOR "/"

#endif

#endif */\* RUN\_INPUT\_TEST \*/*

*// returns true if memory leak*

**bool** testMemoryLeak();

*// helper function to parse index and value from input - with index validation. Returns parse fails or if index < 0.*

**bool** parsePriorityName(**const** string &input, **int** &priority, string &name);

*// runs tests for the project.*

**void** runTests();

*// runs the UI for the project.*

**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;

#ifdef RUN\_INPUT\_TEST

*// Override cin with a test input file.*

string testInputFile(\_\_FILE\_\_);

size\_t filePos = testInputFile.rfind(PATH\_SEPARATOR);

testInputFile = testInputFile.erase(filePos + 1, string::npos);

testInputFile += TEST\_INPUT;

ifstream in(testInputFile.c\_str());

cin.rdbuf(in.rdbuf()); *//redirect std::cin to in.txt!*

#endif */\* RUN\_INPUT\_TEST \*/*

#ifdef RUN\_TESTS

runTests();

#else

runProgram();

#endif */\* RUN\_TESTS \*/*

testMemoryLeak();

**return** 0;

}

*// run tests for the project*

**void** runTests() {

PriorityQueue queue;

Node node;

string name = "George";

queue.penqueue(1, name);

queue.penqueue(1, name);

cout << queue << endl;

queue.clearMaxHeap();

queue.penqueue(1, name);

queue.penqueue(2, name);

cout << queue << endl;

node = queue.pdequeue();

assert(node.mPriority == 2);

node = queue.pdequeue();

assert(node.mPriority == 1);

assert(queue.isEmpty());

queue.clearMaxHeap();

queue.penqueue(1, name);

queue.penqueue(2, name);

queue.penqueue(3, name);

cout << queue << endl;

node = queue.pdequeue();

assert(node.mPriority == 3);

node = queue.pdequeue();

assert(node.mPriority == 2);

node = queue.pdequeue();

assert(node.mPriority == 1);

assert(queue.isEmpty());

queue.clearMaxHeap();

queue.penqueue(1, name);

queue.penqueue(2, name);

queue.penqueue(3, name);

queue.penqueue(4, name);

cout << queue << endl;

node = queue.pdequeue();

assert(node.mPriority == 4);

node = queue.pdequeue();

assert(node.mPriority == 3);

node = queue.pdequeue();

assert(node.mPriority == 2);

node = queue.pdequeue();

assert(node.mPriority == 1);

assert(queue.isEmpty());

}

*// returns true if memory leak*

**bool** testMemoryLeak() {

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

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

**return** **true**;

}

**return** **false**;

}

*// helper function to parse index and value from input - with index validation. Returns parse fails or if index < 0.*

**bool** parsePriorityName(**const** string &input, **int** &priority, string &name) {

stringstream ss;

ss.str(input);

ss >> priority >> name;

**return** !ss.fail() && priority >= 1 && priority <= 10;

}

*// runs the UI for the project.*

**void** runProgram() {

PriorityQueue queue;

string input, name;

**int** priority = 0;

cout << "Please enter data as <priority> <name>, enter '-1' to exit: " << endl;

getline(cin, input);

**while** (input != "-1") {

**if** (parsePriorityName(input, priority, name)) {

queue.penqueue(priority, name);

}

getline(cin, input);

}

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

cout << queue.pdequeue() << endl;

}

}

Employee.h:

*/\**

*Programmer: Jessica Sullivan*

*Programmer's ID: 1282151*

*Class: COMSC-210-5067*

*\*/*

*// PriorityQueue.h*

*// CS210\_Assignment9\_Heaps*

*//*

*// Created by Jessie Sully on 4/17/20.*

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

*//*

#ifndef PriorityQueue\_h

#define PriorityQueue\_h

#include <iostream>

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

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

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

*// Priority Queue Node*

**struct** Node {

*// increments sNumNodeObjects for memory leak check*

Node() { sNumNodeObjects++; }

*// increments sNumNodeObjects for memory leak check*

Node(**const** **int** priority, **const** string &name) : mPriority(priority), mName(name) { sNumNodeObjects++; }

*// decrements sNumNodeObjects for memory leak check*

~Node();

*// copies member variables*

Node& **operator**=(**const** Node& node);

*// makes streaming easier*

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

**int** mPriority = 0;

string mName;

};

Node::~Node() {

sNumNodeObjects--;

}

Node& Node::**operator**=(**const** Node &node) {

mPriority = node.mPriority;

mName = node.mName;

**return** \***this**;

}

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

ostr << node.mPriority << ' ' << node.mName;

**return** ostr;

}

#ifdef RUN\_TESTS

**static** **const** **int** DEFAULT\_SIZE = 2;

#else

**static** **const** **int** DEFAULT\_SIZE = 10;

#endif */\* RUN\_TESTS \*/*

*// Binary Search Tree*

**class** PriorityQueue {

**public**:

PriorityQueue() { mMaxHeap = **new** Node[DEFAULT\_SIZE]; }

*// Runs clearTree to free allocated space*

~PriorityQueue();

*// returns mRoot*

Node\* getRoot() **const** { **return** mMaxHeap; }

**bool** isEmpty() **const** { **return** mSize == 0; }

**void** penqueue(**int** priority, **const** string &mName);

Node pdequeue();

**void** clearMaxHeap();

**friend** ostream& **operator**<<(ostream& ostr, **const** PriorityQueue &queue);

**private**:

**int** mCapacity = DEFAULT\_SIZE;

**int** mSize = 0;

Node\* mMaxHeap = **nullptr**;

**void** increaseCapacity();

**void** copyNodes(Node\* largerArray) **const**;

**void** swapNodes(**int** index1, **int** index2);

**void** reheapify(**int** startIndex);

**void** siftDown(**int** startIndex);

};

PriorityQueue::~PriorityQueue() {

**delete** [] mMaxHeap;

}

**void** PriorityQueue::clearMaxHeap() {

**delete** [] mMaxHeap;

mSize = 0;

mCapacity = DEFAULT\_SIZE;

mMaxHeap = **new** Node[DEFAULT\_SIZE];

}

**void** PriorityQueue::copyNodes(Node\* largerArray) **const** {

**for**(**int** i = 0; i < mSize; i++) {

largerArray[i] = mMaxHeap[i];

}

}

**void** PriorityQueue::increaseCapacity() {

mCapacity += DEFAULT\_SIZE;

Node\* largerArray = **new** Node[mCapacity];

copyNodes(largerArray);

**delete** [] mMaxHeap;

mMaxHeap = largerArray;

}

**void** PriorityQueue::swapNodes(**int** index1, **int** index2) {

Node storageNode;

storageNode = mMaxHeap[index1];

mMaxHeap[index1] = mMaxHeap[index2];

mMaxHeap[index2] = storageNode;

}

**void** PriorityQueue::reheapify(**int** startIndex) {

**for** (**int** i = startIndex, j = (startIndex - 1) / 2; i > 0;

i = j, j = (j - 1) / 2) {

**if** (mMaxHeap[i].mPriority > mMaxHeap[j].mPriority) {

swapNodes(i, j);

}

}

}

**void** PriorityQueue::penqueue(**int** priority, **const** string &name) {

**if** (mSize == mCapacity) {

increaseCapacity();

}

mMaxHeap[mSize].mPriority = priority;

mMaxHeap[mSize].mName = name;

reheapify(mSize);

mSize++;

}

**void** PriorityQueue::siftDown(**int** startIndex) {

**for** (**int** i = startIndex, j = 2 \* startIndex + 1; j < mSize;

i = j, j = 2 \* i + 1) {

**if** (mMaxHeap[j].mPriority < mMaxHeap[j + 1].mPriority ) {

j++;

}

**if** (mMaxHeap[i].mPriority < mMaxHeap[j].mPriority) {

swapNodes(i, j);

}

}

}

Node PriorityQueue::pdequeue() {

Node returnNode;

returnNode = mMaxHeap[0];

mMaxHeap[0] = mMaxHeap[mSize - 1];

mMaxHeap[mSize - 1].mPriority = 0;

mMaxHeap[mSize - 1].mName.clear();

mSize--;

siftDown(0);

**return** returnNode;

}

ostream& **operator**<<(ostream& ostr, **const** PriorityQueue &queue) {

**for** (**int** i = 0; i < queue.mSize; i++) {

ostr << queue.mMaxHeap[i] << endl;

}

**return** ostr;

}

#endif */\* PriorityQueue\_h \*/*

Output:

A screenshot of a cell phone

Description automatically generated