CS300 Fall 2021 Assignment 3, due October 29, Friday 11:59pm

**Extensions of Link-based Stack**

In this assignment, you will extend the link-based implementation of Stack. This Stack will support 3 more additional functions:

1. T max() const;

This function always returns the maximum item value stored in this stack. It is not necessarily the item on the top of the Stack. It can be anywhere in the Stack. Your task is to implement this function efficiently such that it doesn’t need to traverse the whole linked list to find the max item for each call. Or in short, **it runs in constant time**.

1. T mean() const;

Like the above function, this function will return the average (mean) value of all items in this Stack. If the Stack holds integer data type, then this function will return the integer value using integer division. For example, the mean of 1 and 2 is (1+2)/2 = 1, while the mean of 1, 2, and 3 is (1+2+3)/3 = 2. Your task is to implement this function efficiently such that it doesn’t need to traverse the whole linked list to find the mean value. Or in short, **it runs in constant time**.

1. Stack<T> operator+(const Stack<T> &other);

This operator will zip two equal sized Stacks together while not breaking the above two functions. For example, given two Stacks implemented in linked list:

Stack1: 4->3->2->1->NULL

Stack2: 5->6->7->8->NULL

Stack1 + Stack2 will be: 4->5->3->6->2->7->1->8->NULL where the new Stack will start with the top of Stack1, then it inserts the items from Stack2 and Stack1 alternatively like a zipper.

**Implementation and test:**

You will extend the link-based implementation of Stack from the textbook. **Make sure the sample code compiles and works before you proceed to the extensions.** You can either design additional data structure to help you implement your functions or you can create additional member variables and helper functions. **You don’t need to use any data structure or algorithms that we haven’t covered in class.**

There are two files provided: **main.cpp** and **MyRand.h**.

* The MyRand.h file is an implementation of C++ random number generator, so that the results are consistent across all compilers.
* The main.cpp file includes all the test cases and auto-grading, so don’t modify it. It will automatically grade out of 40 points according to the correctness. So, you know exactly what you will get in that portion. Once you submit your assignment, you accept your final score for correctness.
* In the header portion of main.cpp, it #include a **Stack.cpp** and an **Node.cpp** file, which are what you need to implement by yourself in separate files. This is necessary for compiling template classes if the \*.h and \*.cpp are separate files. Alternatively, you can also put both the definition and the implementation of the template class in one \*.h file.

You don’t need to handle other edge cases.

**Submission**:

Put all your code in one folder and do the following sanity check to make sure your code is clean and work correctly.

Windows Users:

yourfolder> g++ -o myProgram.exe -std=c++14 \*.cpp

yourfolder>myProgram.exe

Mac Users:

yourfolder$ g++ -o myProgram -std=c++14 \*.cpp

yourfolder$ ./myProgram

**Submit your folder as a single zip file to Canvas.**

MyRand.h

#ifndef MYRAND\_H\_

#define MYRAND\_H\_

unsigned long int rand\_next = 1;

int my\_rand()

{

rand\_next = rand\_next \* 1103515245 + 12345;

return ((unsigned int)(rand\_next / 65536) % 32768);

}

void my\_srand(unsigned int seed) { rand\_next = seed; }

#endif

main.cpp

#include <iostream>

#include "MyRand.h"

#include "Stack.cpp"

#include "Node.cpp"

using namespace std;

int score = 0;

void grade(bool condition, int points)

{

if (condition)

{

cout << "Pass" << endl;

score += points;

}

else

{

cout << "Fail" << endl;

}

}

void test1()

{

int points = 0;

StackInterface<int> \*stackPtr1 = new Stack<int>();

stackPtr1->push(1);

grade(stackPtr1->max() == 1, 1);

grade(stackPtr1->mean() == 1, 1);

stackPtr1->push(2);

grade(stackPtr1->max() == 2, 1);

grade(stackPtr1->mean() == 1, 2);

stackPtr1->push(3);

grade(stackPtr1->max() == 3, 1);

grade(stackPtr1->mean() == 2, 2);

delete stackPtr1;

}

void test2()

{

my\_srand(0);

StackInterface<int> \*stackPtr = new Stack<int>();

for (int i = 0; i < 1000; i++)

stackPtr->push(my\_rand() % 100 + i);

int maxes[10] = {1020, 924, 924, 826, 808, 762, 734, 706, 688, 589};

int means[10] = {513, 468, 464, 416, 408, 386, 379, 359, 353, 303};

for (int i = 0; i < 10; i++)

{

int popCounts = my\_rand() % 100;

for (int j = 0; j < popCounts; j++)

{

stackPtr->pop();

}

grade(maxes[i] == stackPtr->max(), 1);

grade(means[i] == stackPtr->mean(), 1);

}

delete stackPtr;

}

void test3()

{

Stack<int> stack1;

Stack<int> stack2;

stack1.push(4);

stack1.push(2);

stack1.push(6);

stack1.push(10);

stack1.push(8);

stack2.push(5);

stack2.push(1);

stack2.push(3);

stack2.push(9);

stack2.push(7);

Stack<int> stack = stack1 + stack2;

stack.pop();

grade(stack.max() == 10 && stack.mean() == 5, 1);

for (int i = 0; i < 5; i++)

stack.pop();

grade(stack.max() == 5 && stack.mean() == 3, 1);

}

void test4()

{

my\_srand(0);

Stack<int> stack1;

Stack<int> stack2;

for (int i = 0; i < 1000; i++)

{

stack1.push(my\_rand() % 100 + i);

stack2.push(my\_rand() % 100 + i / 2);

}

Stack<int> result = stack1 + stack2;

int maxes1[5] = {1044, 949, 893, 830, 759};

int means1[5] = {523, 480, 450, 433, 393};

int maxes2[5] = {563, 523, 491, 468, 439};

int means2[5] = {285, 263, 248, 240, 219};

int maxes[5] = {1066, 1020, 990, 961, 927};

int means[5] = {414, 397, 386, 380, 364};

for (int i = 0; i < 5; i++)

{

int popCounts = my\_rand() % 100;

for (int j = 0; j < popCounts; j++)

{

result.pop();

stack1.pop();

stack2.pop();

}

grade(stack1.max() == maxes1[i] &&

stack1.mean() == means1[i] &&

stack2.max() == maxes2[i] &&

stack2.mean() == means2[i],

1);

grade(maxes[i] == result.max() &&

means[i] == result.mean(),

1);

}

}

int main()

{

try

{

test1();

test2();

test3();

test4();

}

catch (exception e)

{

cout << e.what() << endl;

}

cout << "Your total correctness score is: " << score << endl;

return 0;

}