**CSCI 241 Assignment 7  
100 points**

**Purpose**

This assignment is an exercise in implementing the stack ADT using a singly-linked list. This assignment also introduces the concept of templates.

**Assignment**

This program creates and implements a class to represent the Stack ADT using a singly-linked list.

A *driver program* is provided for this assignment to test your implementation. You don't have to write the tests.

**Program**

You will need to write one template structure and one template class for this assignment, called Node and Stack. You will need to implement several methods and functions associated with these data types.

**Since these are both C++ templates and are closely related to each other, all of your code should be placed in a single header (.h) file. This includes the implementations of all methods and any other associated functions.**

**struct Node**

*Data members*

This template structure should have two data members: a member of the template parameter type to store an item to be inserted into the stack, and a pointer to a Node. This pointer, next, will point to the next node in the linked list (or be nullptr if this is the last node in the list).

Since the Stack class will need access to these data members, make them public (which is the default for a struct).

*Methods*

* Constructor

The structure should have a constructor that can be used to initialize the data members of the stack node.

**class Stack**

*Data members*

This class should have two data members. The first is a pointer to an Node. This pointer, stkTop, will point to the first node in the list, i.e. the top node in the stack (or will be nullptr if the stack is empty). The second data member should be an size\_t or unsigned integer that will be used to store the *stack size*, the number of items currently stored in the stack.

*Methods and associated functions*

* Constructor

The class should have a default constructor that takes no arguments. The constructor should set stkTop to nullptr and set the stack size to 0.

* Destructor

The class should have a destructor. The destructor can simply call the clear() method described below.

* Copy constructor

The class should also have a proper copy constructor.

* operator=

The assignment operator should be properly overloaded.

* operator<<

The output operator should be overloaded so that an entire Stack can be sent to the standard output. As usual, this function will need to be a friend rather than a method. Declaring a template function to be a friend of a template class requires some special syntax - see the **Implementation Hints** below.

* clear()

This method takes no arguments and returns nothing. It should properly set the stack back to the empty state. That means deleting all of the nodes in the stack, setting the top pointer back to nullptr, and setting the stack size back to 0. One easy way to accomplish this is to repeatedly pop until the stack is empty.

* size()

This method takes no arguments and returns an unsigned integer or size\_t. It should return the stack size; i.e., the number of data items currently stored in the stack.

* empty()

Returns true if there are no data items currently stored in the stack; otherwise returns false.

* top()

This method takes no arguments and returns a reference to a constant item of the template parameter type. It should return the data stored in the top node of the stack (i.e., the first node in the linked list). You may assume this method will **not** be called if the stack is empty.

* push()

This method takes a reference to a constant item of the template parameter type as its argument (the item to insert into the stack). It returns nothing. The method should insert the item at the top of the stack (at the front of the linked list).

* pop()

This method takes no arguments and returns nothing. It should remove the node at the top of the stack. You may assume this method will **not** be called if the stack is empty.

If you like, you may write private methods for the Stack class in addition to the methods described above. For example, you may want to write a copyList() method that can be called by both the copy constructor and overloaded assignment operator to make a copy of the linked list.

**Driver Program**

A driver program, assign7.cpp is provided for this assignment. The purpose of a driver program is to test other pieces that you code. You do not need to write the driver program yourself. A copy of the driver program can also be found on turing at /home/turing/t90kjm1/CS241/Code/Fall2017/Assign7/assign7.cpp.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PROGRAM: CSCI 241 Assignment 7

PROGRAMMER: your name

LOGON ID: your z-ID

DUE DATE: due date of assignment

FUNCTION: This program tests the functionality of the Stack

template class.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "Stack.h"

#include <iostream>

using std::cout;

using std::endl;

int main()

{

cout << "Testing default constructor\n\n";

Stack<int> s1;

cout << "s1 (size " << s1.size() << "): " << s1 << endl;

cout << "s1 is " << ((s1.empty()) ? "empty\n" : "not empty\n");

cout << endl;

cout << "Testing push()\n\n";

s1.push(17);

cout << "s1 (size " << s1.size() << "): " << s1 << endl;

cout << "s1 is " << ((s1.empty()) ? "empty\n" : "not empty\n");

cout << endl;

s1.push(2);

s1.push(6);

s1.push(4);

cout << "s1 (size " << s1.size() << "): " << s1 << endl;

cout << "s1 is " << ((s1.empty()) ? "empty\n" : "not empty\n");

cout << endl;

cout << "Testing copy constructor\n\n";

Stack<int> s2(s1);

cout << "s1 (size " << s1.size() << "): " << s1 << endl;

cout << "s2 (size " << s2.size() << "): " << s2 << endl << endl;

cout << "Testing clear()\n\n";

s1.clear();

cout << "s1 (size " << s1.size() << "): " << s1 << endl;

cout << "s2 (size " << s2.size() << "): " << s2 << endl << endl;

Stack<int> s3;

s3.push(36);

s3.push(41);

s3.push(75);

s3.push(28);

cout << "s3 (size " << s3.size() << "): " << s3 << endl << endl;

cout << "Testing assignment operator\n\n";

Stack<int> s4;

s4 = s3;

cout << "s3 (size " << s3.size() << "): " << s3 << endl;

cout << "s4 (size " << s4.size() << "): " << s4 << endl << endl;

s3.clear();

cout << "s3 (size " << s3.size() << "): " << s3 << endl;

cout << "s4 (size " << s4.size() << "): " << s4 << endl << endl;

cout << "Testing assignment to self\n\n";

s4 = s4;

s3 = s4;

s4.clear();

cout << "s3 (size " << s3.size() << "): " << s3 << endl;

cout << "s4 (size " << s4.size() << "): " << s4 << endl << endl;

cout << "Testing chained assignment\n\n";

Stack<int> s5;

s5 = s4 = s3;

cout << "s3 (size " << s3.size() << "): " << s3 << endl;

cout << "s4 (size " << s4.size() << "): " << s4 << endl;

cout << "s5 (size " << s5.size() << "): " << s5 << endl << endl;

cout << "Testing top(), push(), pop()\n\n";

Stack<char> s6, s7;

for(char c = 'a'; c < 'k'; c++)

s6.push(c);

cout << "s6 (size " << s6.size() << "): " << s6 << endl << endl;

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

{

int val;

val = s6.top();

s7.push(val);

s6.pop();

}

cout << "s6 (size " << s6.size() << "): " << s6 << endl;

cout << "s7 (size " << s7.size() << "): " << s7 << endl << endl;

cout << "Testing top()\n";

s6 = s7;

int val1 = s6.top();

int val2 = s7.top();

val1 = s6.top(); // Make sure that top() doesn't remove a value.

cout << ((val1 == val2) ? "top() works\n\n" : "top() failure\n\n");

cout << "Testing const correctness\n\n";

s7.clear();

const Stack<char>& r6 = s6;

cout << "s6 (size " << r6.size() << "): " << r6 << endl;

cout << "s6 is " << ((r6.empty()) ? "empty\n" : "not empty\n");

cout << "Top item of s6: " << r6.top() << endl << endl;

s7 = r6;

Stack<char> s8(r6);

cout << "s7 (size " << s7.size() << "): " << s7 << endl;

cout << "s8 (size " << s8.size() << "): " << s8 << endl << endl;

return 0;

}

**Output**

Output from the correctly functioning driver program should look like the following:

Testing default constructor

s1 (size 0):

s1 is empty

Testing push()

s1 (size 1): 17

s1 is not empty

s1 (size 4): 4 6 2 17

s1 is not empty

Testing copy constructor

s1 (size 4): 4 6 2 17

s2 (size 4): 4 6 2 17

Testing clear()

s1 (size 0):

s2 (size 4): 4 6 2 17

s3 (size 4): 28 75 41 36

Testing assignment operator

s3 (size 4): 28 75 41 36

s4 (size 4): 28 75 41 36

s3 (size 0):

s4 (size 4): 28 75 41 36

Testing assignment to self

s3 (size 4): 28 75 41 36

s4 (size 0):

Testing chained assignment

s3 (size 4): 28 75 41 36

s4 (size 4): 28 75 41 36

s5 (size 4): 28 75 41 36

Testing top(), push(), pop()

s6 (size 10): j i h g f e d c b a

s6 (size 0):

s7 (size 10): a b c d e f g h i j

Testing top()

top() works

Testing const correctness

s6 (size 10): a b c d e f g h i j

s6 is not empty

Top item of s6: a

s7 (size 10): a b c d e f g h i j

s8 (size 10): a b c d e f g h i j

**Implementation Hints**

* Implement this similarly to the last assignment. Start off at the beginning of the driver program and try to get it working piece by piece. Maintain a working program as you go.
* Declaring a template function to be a friend of a template class is one of the classic "gotcha's" in C++. We are trying to declare a function to be a friend of **all** classes that might be instantiated from the Stack template class, and most C++ compilers will quite properly refuse to do that without some special syntax.

The friend declaration must contain an extra set of <> to indicate that it is a template function (however, do not code this in the actual function definition - only the friend declaration). You'll also usually need to *forward declare* both the template class and the template function, as shown below.

How much of this you actually need to do can vary from compiler to compiler, but the code shown below should work in Quincy, Dev-C++, and with g++ on turing/hopper.

#ifndef STACK\_H

#define STACK\_H

#include <iostream>

template <class T>

struct Node

{

...

};

// Method definitions for the Node class

// Forward declaration of the Stack template class

template <class T>

class Stack;

// Forward declaration of the operator<< template function

template <class T>

std::ostream& operator<<(std::ostream&, const Stack<T>&);

template <class T>

class Stack

{

// friend declaration for the template function - note the

// special syntax

friend std::ostream& operator<< <>(std::ostream&, const Stack<T>&);

...

};

// Method definitions for the Stack class

#endif /\* STACK\_H \*/

**Other Points**

* A makefile is required. Same as always. Make sure it has appropriate rules for all the pieces involved. Obviously, with only one .cpp file (assign7.cpp), there won't be very many rules to write! **Remember, there is no Stack.cpp for this assignment, only Stack.h.**
* Don't try to write a makefile rule to compile your Stack.h header file. Header files are compiled automatically when the source file they're included in gets compiled. Compiling a header file directly will produce something called a *pre-compiled header file*, a huge file with the extension .gch. That's not what you want for this assignment.
* Note that the driver program assumes that your header file is called Stack.h. If you name your header file differently, you'll need to modify the driver program accordingly.
* Programs that do not compile on turing/hopper automatically receive 0 points.
* Submit your program using the electronic submission guidelines posted on the course web site.