**CSE 330 Lab 6 Report**

Daniel Meyer

Data Structures

Fall 2017

**Status:** 100%

**Time Complexity:** O(n)

**Storage Complexity:** O(n)

**Source Code:** Stack.h – Pages 2-3

Queue.h – Pages 4-5

Queue\_test.cpp – Page 6

**Sample Run:**

/\*

Daniel Meyer

10-31-17

CSE 330

Fall 2017

Lab 6: Stack and Queue

Problem: Implement classes for "Stack" and "Queue" that have the functionality of their STL

counterparts but as template and container classes.

Algorithm: Using a default container of List from Lab 5, Stack contains functions for empty,

size, push, pop, and top. These functions are implemented using functions provided by the

container, i.e. List's push\_back() and pop\_back().

Status: 100%

Time Complexity: O(n) Storage Complexity: O(n)

\*/

#ifndef STACK\_H

#define STACK\_H

// Stack.h -- a stack implemented as an adapter (of vector or list or ...)

#include "../Lab5/List.h"

using namespace std;

//Use the following line for STL containers.

//template <class T, template <class T, class = allocator<T> > class Container = list>

template <class T, template <class T> class Container = List>

class Stack

{

public:

//We don't need a constructor or destructor because the Container has/should have one

//Stack(): container() { }

//~Stack() { ~container(); }

bool empty() const;

unsigned int size() const;

void push(const T & x);

void pop();

T & top();

private:

Container<T> container;

};

//Checks if the Stack is empty

template <class T, template <class T> class Container = List>

bool Stack<T, Container>::empty() const { return container.empty(); }

//Returns the number of elements in the Stack

template <class T, template <class T> class Container = List>

unsigned int Stack<T, Container>::size() const { return container.size(); }

//Pushes (back) an element on the Stack

template <class T, template <class T> class Container = List>

void Stack<T, Container>::push(const T &x) { container.push\_back(x); }

//Pops (back) element from the Stack

template <class T, template <class T> class Container = List>

void Stack<T, Container>::pop() { container.pop\_back(); }

//Returns value at the back of the Stack without removing it

template <class T, template <class T> class Container = List>

T& Stack<T, Container>::top() { return container.back(); }

#endif

/\*

Daniel Meyer

10-31-17

CSE 330

Fall 2017

Lab 6: Stack and Queue

Problem: Implement classes for "Stack" and "Queue" that have the functionality of their STL

counterparts but as template and container classes.

Algorithm: Using a default container of List from Lab 5, Queue contains functions for empty,

size, push, pop, and top. These functions are implemented using functions provided by the

container, i.e. List's push\_back() and pop\_front().

Status: 100%

Time Complexity: O(n) Storage Complexity: O(n)

\*/

#ifndef QUEUE\_H

#define QUEUE\_H

#include "../Lab5/List.h"

using namespace std;

//Queue class with template and container type (List default type) and functions emulating FIFO Queue

template <class T, template <class T> class Container = List>

class Queue

{

public:

bool empty() const;

unsigned int size() const;

void push(const T & x);

void pop();

T & top();

private:

Container<T> container;

};

//Fuctions checking if the Queue is empty

template <class T, template <class T> class Container = List>

bool Queue<T, Container>::empty() const { return container.empty(); }

//Returns number of elements in the Queues

template <class T, template <class T> class Container = List>

unsigned int Queue<T, Container>::size() const { return container.size(); }

//Pushes (back) an element onto the Queue

template <class T, template <class T> class Container = List>

void Queue<T, Container>::push(const T &x) { container.push\_back(x); }

//Pops (front) element from Queue

template <class T, template <class T> class Container = List>

void Queue<T, Container>::pop() { container.pop\_front(); }

//Returns value at the front of the Queue without removing it

template <class T, template <class T> class Container = List>

T& Queue<T, Container>::top() { return container.front(); }

#endif

/\*

Daniel Meyer

10-31-17

CSE 330

Fall 2017

Lab 6: Stack and Queue

Problem: Implement classes for "Stack" and "Queue" that have the functionality of their STL

counterparts but as template and container classes.

Algorithm:

Status: 100%

Time Complexity: O(n) Storage Complexity: O(n)

\*/

#include <iostream>

#include <cassert>

#include "Queue.h"

#include "../Lab3/String.h"

#include "../Lab5/List.h"

using namespace std;

//Test class using instances of Queue as an int and double with default List container,

//as well as an instance using the String class from Lab 3 with the default List container.

int main()

{

Queue<int> i; // uses List as the default container

i.push(5);

i.push(6);

cout << i.top() << endl;

Queue<String, List> s; // uses Vector as the container

s.push("abc");

s.push("de");

s.pop();

cout << s.top() << endl;

Queue<double> d;

d.push(123.45);

d.push(78.90);

d.push(1.2);

d.push(3.4);

int size = d.size();

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

{

cout << d.top() << endl;

d.pop();

}

return 0;

}