Project 1 – Linked List Stack and Queue

Version:
Initial: 21 January 2016
Release: 12 February 2016

Due: Please check due date on BlackBoard

Objectives

The objective of this programming assignment are:

- Practice using templates and inheritance.
- Implement a data structure (Linked Lists)
- Use the data structure to mimic and enhance STL data structures and implement Abstract Data Types (Stacks and Queues)
- Practice error handling.
- Write code to a specification that someone else will use.

Introduction

C++ includes a Standard Template Library (STL) that contains pre-built code for many data structures. Examples of these data structures would be linked lists, stacks, queues, double ended queues, pairs, vectors, etc. Using STL containers allows for quick implementation of data structures as well as increasing portability of code. Each data structure has its own interface that allows access to the methods and storage, standardizing their use.

STL containers do have shortcomings and one is error handling. STL containers expect you, as the programmer, to protect your code and not induce errors. This includes doing things like asking for the next item, using Top() on an **empty** stack. (Nothing there!!)

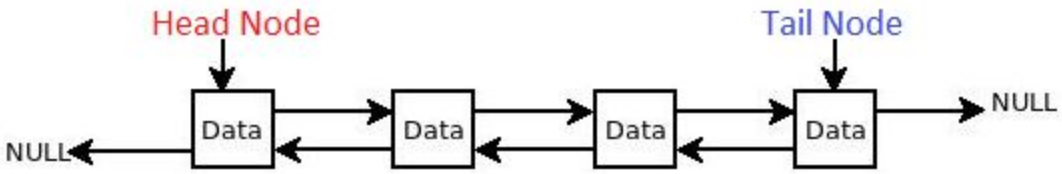
This programming project will replicate two of the STL containers (stacks and queues) and enhance them to protect against ***common errors*** in use.

This project is also designed for you to program to a specification. This means you will write a data structure based on the included specification but you will not implement this into a larger part of the project. At times other people will use the code you write and you will not

have the ability to fix it along the way. These means you need to write solid code to the specification and ensure that code works via testing.

Linked List Structure

There are different ways to build a linked list: singly, doubly, with or without dummy nodes, internal and external to the data etc. For this project a doubly linked list without dummy nodes is required. This means the linked list will be traversable either forward or backwards and that the head and tail nodes will contain data. There will be no empty head or tail nodes. **You will construct this type of Linked List.**



This would represent a 4 node linked list. The right arrows are for the "next" node and the left arrows are for the "previous" node.

Class Structure

You will implement four classes for this project:

- Node341 – This is the linked list node.
- List341 – This is the parent container.
- Stack341 – This a child and implements the stack.
- Queue341 – This a child and implements the queue.

In addition to these classes you will be given a class called Exceptions341. This will allow you to capture and pass exceptions.

Details on each class will follow, as well as a diagram showing how classes fit together.

Node341 Class (Node341.h and Node341.cpp)

This class is designed to hold the data that is being stored, as well as pointers to the next and previous nodes.

It must be(use) a C++ template.

List341 Class (List341.h and List341.cpp)

This class is the parent Linked List class and must be a C++ template. It should be defined as a pure virtual (abstract) class. This means that it can not be instantiated as a stand alone object.

List341 will hold most of the data members and methods for the child classes, but will have almost no code of its own. The only implementation code in this class will be in the constructor and, possibly, the destructor.

Scope

For this class, methods will either be public (and be an interface in the child classes) or protected (and inherited by the children but not available outside of the class). All data members that are inherited should be protected and if they are not inherited should be private. There must not be any public data members.

Methods

Along with appropriate constructors and destructor this class must present the following methods:

```
virtual int  Size() const = 0;          \\Returns the size of a list
virtual bool Empty() const = 0;  \\Returns whether a list has data or not
virtual bool Push(T obj) = 0;           \\Pushes data into the list
virtual bool Pop() = 0;                 \\Pops data from a list
virtual bool Clear() = 0;      \\Clears a list
```

Stack341 Class (Stack341.h and Stack341.cpp)

This class will be a child class for List341 and use public access control. It must be a C++ template.

The virtual methods inherited will need to be defined in this class' implementation file. The following will also need to be defined and implemented:

```
T Top(); //Returns the top item on the stack
```

Queue341 Class (Queue341.h and Queue341.cpp)

This class will be a child class for List341 and use public access control. It must be a C++ template.

The virtual methods inherited will need to be defined in this class' implementation file. The following will also need to be defined and implemented:

```
T Front(); //Returns the item at the front of the queue
T Back();  //Returns the item at the back of the queue
```

Exceptions341 Class (Exceptions341.h)

This class is provided and must be used. To use the provided class you simply create the object with a string that describes the exception that is being raised.

Example:

```
throw Exceptions341("Stack – Could not determine if the stack is full")
catch (Exceptions341 &E){
        cout <<  E.GetMessage << endl;
}
```

This is a guide and not necessarily a real world example.

Errors and Exceptions

Misuse of the stack and queue implementations in C++ STL can cause segmentation faults. The 341 implementations should not cause segmentation faults, instead they should return exceptions when warranted that can be gracefully handled. The following table contains the return types and exceptions that are expected:

Method	Success	Failure
Top()	T[1]	exception[2]

Front()	T[1]	exception[2]
Back()	T[1]	exception[2]
Clear()	true	false[3]
Pop()	true	false
Push()	true	false

[1] This is the item that has been stored.
[2] This is an exception object, see example above.
[3] A Failure means an error occurred. An empty stack or queue can be cleared without error.

STL Containers

This project is designed to implement a stack and queue using a custom made linked list.
Use of STL containers to complete this project will result in a score of zero (0).

You may use STL containers to understand the functionality involved in stacks, queues and linked lists. It is encouraged that you write your own test code to understand how the different data structures function, but you should NOT turn in any test code, especially if it uses any STL container.

Driver (Driver.cpp)

A sample driver will be supplied on BlackBoard. This driver will illustrate the format we will use when testing your project, though it is not exhaustive of what will be tested.

You must submit a test driver for the graders. The graders will also have a driver to test with as well. This test driver should show you took time to test your code before submission and should be representative of what you think are good tests of your project.

What to Submit

Follow the course project submission procedures. You should copy over all of your C++ source code with .cpp/.h files under the src directory. You must also supply a Makefile build file.
Make sure that your code is in the ~/cs341proj/proj1/src directory and not in ~/cs341proj/proj1/. In particular, the following Unix commands should work.

```
cd ~/cs341proj/proj1/src
make
make run
make clean
```

The command “make run” should simply run the project that compiled successfully.

Don't forget the Project Submission requirements shown online!! One hint, **after you submit**, if you type:

```
ls ~/cs341proj/proj1/
```

and you see a bunch of .cpp and .h files, this is WRONG. You should see:

```
src
```

instead. The C++ programs must be in directories under the src directory. Your submissions will be compiled by a script. The script will go to your proj1 directory and run your makefile. This is required. You will be severely penalized if you do not follow the submission instructions.

Links

You may find the following links helpful

http://www.cplusplus.com/reference/queue/queue/

http://www.cplusplus.com/reference/stack/stack/

http://www.tutorialspoint.com/cplusplus/cpp_inheritance.htm

http://www.prenhall.com/divisions/esm/app/kafura/secure/chapter7/html/7.5_inheritance.htm

Addendum

None yet!