A4: Stack

Overview

You are tasked with creating a basic Stack using nodes with links going forwards and backwards (i.e., a doubly linked list).

Deliverables

- **A4.tar** will be submitted to A4 assignment on Autolab.
    - This will contain your Stack.hpp and Stack.cpp files
    - All other files will be overwritten, so ensure all your code is contained in your Stack.hpp and Stack.cpp files. Make sure they are not within any folders.

Due Date

DUE DATE:

A4.tar: 4/3/2017, at 17:00 (this is Monday, April 3 at 5:00PM).

***** Please be aware of this deadline. Autolab will also tell you how long until the deadline.*****

Start Early!!!!!!!!

Late Policy

The policy for late submissions is as follows:

- Before the deadline: 100% of what you earn (from your best submission).
- One day late: 25-point penalty (lower your earned points by 25, minimum of 0).
- > 1 days late: 0 points.

Keep track of the time if you are working up until the deadline. Submissions become late after the set deadline. **Please note the change in submission time.**

Objectives

In this assignment, you will create a Stack implemented using a doubly linked list.

Useful Resources

Review the lecture notes and provided examples to get an idea for using lists and stack functionality.

Instructions

- Download the file `A4-skeleton.tar` and extract the skeleton files. **Rename the skeleton files appropriately.**
- A `CMakeLists.txt` file is included to properly build your program. You may use this by Importing the folder into CLion, or from the command line by the following commands:

```
cmake .
```

```
make
```

- You will have to add the function declarations in `Stack.hpp` and the definitions in `Stack.cpp`. You may add any member variables or helper methods/functions you wish add to the `Stack.hpp` file. All function/method definitions should be in `Stack.cpp`.
- **You may not use any other standard libraries within your code aside from <stdexcept> within your Stack.cpp or Stack.hpp files**.

Required:

Complete the functionality for the Stack. You will lose credit if your code leaks memory.

Stack Class:

```
    /**
     * Constructor should correctly initialize any member variables.
     *
     */
    Stack();
```

This constructor should initialize any member variables you wish to use.

```
    /**
     * Copy constructor:
     *    Create a deep copy of the other Stack.
     * @param other - other Stack to copy from.
     */
    Stack(const Stack& other);
```

This constructor should perform a deep copy of the other object (see lecture notes).

```
    /**
     * Copy assignment operator:
     *    Create a deep copy of the other Stack.
     * @param other - other Stack to copy from.
     * @return reference to this.
     */
    Stack& operator=(const Stack& other);
```

This operator should perform a deep copy of the other object into this object (see lecture notes).

```
    /**
     * Destructor:
     *    Clean up all memory used.
     */
    ~Stack();
```

You must clean up any memory used/managed by this data structure. Memory leaks will result in a lower score.

```
    /**
     * push:
     *    Insert the value to the top of the Stack.
     *
     * @param Entry value – value of type Entry to add to Stack.
     */
    void push (const Entry& value);
```

The **push** method will take as input a value of type Entry. This value should be stored on the top of the stack.

```
    /**
     * pop:
     *    Remove the value to the top of the Stack and return a copy.
     *    This function should throw an error if the Stack is empty.
     *
     * @return Entry value – value of type Entry that was removed from Stack.
     */
    Entry pop ();
```

The **pop** method should remove the top value from the Stack and return a copy. Should throw the exact std::runtime_error: "Pop called on empty stack."

```
    /**
     * peek:
     *    Returns a copy of the value on the top of the Stack.
     *    This function should throw an error if the Stack is empty.
     *
     * @return Entry value – value of type Entry that was on top of Stack.
     */
    Entry peek () const;
```

The **peek** method should return a copy of the value on the top of the Stack. Should throw the exact std::runtime_error: "Peek called on empty stack."

```
    /**
     * isEmpty:
     *    Returns whether or not the Stack is empty.
     *
     * @return true: no values are stored in the Stack.
     *         false: one or more values are stored in the Stack.
     */
    bool isEmpty () const;
```

The **isEmpty** method should return true if the Stack is empty and false if there are values stored.

Testing

To test your code, you should write tests in the main function in **driver.cpp**. The default code in the main shows how to throw an std::runtime_error.

Submission

Code Submission (50 points):

You must submit your A4.tar file to the A4 assignment on Autolab.

- Make sure you test extraction of the files prior to submission to ensure all files are present.
- Your submission must compile and run on Autolab. I suggest that you compile often as you build your code to avoid difficulties debugging syntax and other errors.
- Make sure your code does not leak memory.

Tests will be run on your file and your score will be reported to you when finished. For this assignment there is not limit on the number of submissions you may perform. That said, you should not be using the submission system to test and debug your code.

DUE DATE:

A4.tar: 4/3/2017, at 17:00 (this is Monday, April 3 at 5:00PM).

***** Please be aware of this deadline. Autolab will also tell you how long until the deadline.*****