Assignment 08: Linked Lists

COSC 2336: Data Structures and Algorithms

Fall 2020

Objectives

- More practice with dynamic memory and pointers.
- Also more practice with using templates, as we would like our `LinkedList` to be templatized to work with lists of any type.
- Look at a more realistic example of an ADT.
- Learn how to construct linked lists, and perform operations like iterate over, insert and delete items in the list.

Description

In this assignment, you will be given an already templatized version of a LinkedList class. The implementation given is mostly the same one as described and presented in chapter 17 of our textbook. you will be adding some additional operations to the template `LinkedList` class for this assignment.

In particular, you will be adding functionality to use the linked list to search for items, delete items from the list, and display the list, among others. You will need to look at and understand the `LinkedList` class given, and the node data structure(s) that are being managed to add and remove items from the internal linked list.

Setup

For this assignment you will be given the following files:

File Name	Description
`assg08-tests.cpp`	Unit tests for the member functions you are to write.
`LinkedList.hpp`	Header file defining `LinkedList` class and where member function prototypes should be declared.
`LinkedList.cpp`	Implementation file for the `LinkedList` class member functions.

Set up a multi-file project to compile the `.cpp` source files and run them as shown for the class. The Makefile you were given should be usable to create a build project using the Atom editor as required in this class. Since `LinkedList` is a template class you do not actually compile the `LinkedList.cpp` file separately into an object file. If you look at the bottom of the `LinkedList.hpp` header, it includes `LinkedList.cpp`, the template implementations. So any file that includes the `LinkedList.hpp` header file actually also includes the template implementations as well.

The general approach you should take for this assignment, and all assignment is:

1. Set up your project with the given starting code. The files should compile and run, but either no tests will be run, or tests will run but be failing.
2. For this project, start by uncommenting the first `TEST_CASE` in the `assg08-tests.cpp` file. These are the unit tests to test the functionality of `search()` member function, the first member function you are to implement.
3. Add the correct function prototype for the `search()` member function to the `LinkedList` class in the

`LinkedList.hpp` header file. The prototype consists of the name of the member function, its input parameters and their types, and the return value of the function.

4. Add a implementation of the `search()` member function to the `LinkedList.cpp` implementation file. The function should have the same signature as the prototype you gave in the header file. Documentation for the function has not been given for you this time, so add documentation of your function first. Don't forget to indicate that this function is a member of the `ListType` class. And in this assignment, you need to indicate the function is a template function that parameterizes a .
5. Your code should compile and run now. Make sure after adding the function prototype and stub your code compiles and runs. However, your unit tests might be failing initially.
6. Incrementally implement the functionality of your `search()` member function. You should try to add no more than 2 or 3 lines of code, and then make sure your program still compiles and runs. Start by adding code to get the first failing test to pass. Then once that test passes, move on to the next failing tests until you have all tests passing. If you write something that causes a previously passing test to fail, you should stop and figure out why, and either fix it so that the original test still passes, or remove what you did and try a new approach.
7. Once you have the `search()` member function implemented and all unit tests passing, you should then move on to the other functions in the order suggested. Some member functions use previous ones in this assignment, so do them in the order given for you in the tasks below.

I have given you a starting template for your `LinkedList` that implements a lot of the basic functionality of the `LinkedList` class. This code is based on the example `LinkedList` class from our Chapter 17 Malik textbook.

Tasks

You should set up your project/code as described in the previous section. In this section we give some more details on implementing the member functions for this assignment. You should perform the following tasks for this assignment:

1. Write the member function to `search()` the linked list for a particular piece of information. The search function takes a const T& as its parameter and it returns a boolean result. This member function should also be declared as a `const` member function, as it does not change the list if it is called. An example implementaiton for this function is actually given in our textbook, though you may need to change it slightly to work with our assignment code.

2. Also add/write the `deleteNode()` member function, which is also given in our textbook implementation. This function takes a `const T&` as its parameter, which is the value of an item to search for and delete. Thus the first part of this function is similar to the `search()` function, in that you first have to perform a search to find the item. But once found, the node containing the item should be removed from the list (and you should free the memory of the node). This function is only guaranteed to find the first instance of the item and delete it, if the item appears multiple times in the list, the ones after the first one will still be there after the first item is removed by this function. This function should return a `LinkedListItemNotFoundException` if the item asked for is not found. The `LinkedListItemNotFoundException` class has already been defined for you in the starting template header file.
3. Write a member function named `findItemAtIndex()` for the `LinkedList` class. This function will be given a single integer parameter called index. It will search through the linked list and return a reference to the info of index'th node in the list. It is important that you return a T& (a reference to a type T) from this function. This function works using 0 based indexing (like arrays), thus if we ask for index 0, the first or head node info should be returned. If we ask for index 1, the info in the node after the head node is returned. If the list only has 5 nodes (indexes 0 to 4) and we ask for index 5 or greater, you should throw a `LinkedListItemNotFound` exception. (This exception class has already been defined in the `LinkedList` header given to you, you simply need to throw it). For a little extra credit, you can add the overloaded `operator[]` to define indexing operations on your LinkedList which just uses your working `findItemAtIndex()` member function.
4. Write a member function named `deleteItemAtIndex()`. This function will perform similar to the previous one, but instead of returning the info in the index'thed node, it will simply delete the node. Thus your logic will be similar to task 3, you will need to search till you get to the index'thed node in the list. But at that point you should remove the node (and don't forget to delete it, to free up its memory). If the asked for index does not exist, as usual you should throw a `LinkedListItemNotFound` exception.
5. **Extra credit**: you can write this recursive member function for a little bit of additional extra credit. Write a member function named `toReverseString()`. There is already a string function, that creates a string representation of the items in the list and returns the string. Your method will create a string of the items in

the linked list, but in reverse order. You should use the `recursiveReversePrint()` discussed in our textbook as an example. E.g. this method should be implemented using a recursive function definition to accomplish reversing the list. But for credit, your function needs to use recursion, and it needs to build and return a string recursively (not print the results on the `cout` stream). There are tests for this extra credit in the testing file, but you can simply leave them commented out if you don't work on this function. **Hint**: You will need to write two functions named `toReverseString()`. One of them will take no parameters, and is what is usually called by a user of the `LinkedList` class. But the second version should be the recursive function, and it will take a `Node<T>*` as its parameters.

Example Output

Here is the correct output you should get from your program if you correctly implement all the class functions and successfully pass all of the unit tests given for this assignment. If you invoke your function with no command line arguments, only failing tests are usually shown by default. In the second example, we use the -s command line option to have the unit test framework show both successful and failing tests, and thus we get reports of all of the successfully passing tests as well on the output.

```
$ ./test
===============================================================================
All tests passed (168 assertions in 7 test cases)
```

```
$ ./test -s

-------------------------------------------------------------------------------
test is a Catch v2.7.2 host application.
Run with -? for options

-------------------------------------------------------------------------------
<basic functions> test LinkedList basic member function
-------------------------------------------------------------------------------
assg08-tests.cpp:25
.....................................................................

assg08-tests.cpp:31: PASSED:
  CHECK( list.isEmpty() )
with expansion:
  true

... output snipped ...

===============================================================================
All tests passed (168 assertions in 7 test cases)
```

Assignment Submission

A MyLeoOnline submission folder has been created for this assignment. There is a target named `submit` that will create a tared and gziped file named `assg02.tar.gz`. You should do a `make submit` when finished and upload your resulting gzip file to the MyLeoOnline Submission folder for this assignment.

```
$ make submit
tar cvfz assg08.tar.gz assg08-tests.cpp assg08-main.cpp
  LinkedList.hpp LinkedList.cpp
assg08-tests.cpp
assg08-main.cpp
```

`LinkedList.hpp`
`LinkedList.cpp`

Requirements and Grading Rubrics

Program Execution, Output and Functional Requirements

1. Your program must compile, run and produce some sort of output to be graded. 0 if not satisfied.
2. (15 pts.) `search()` works and correctly returns boolean result.
3. (20 pts.) `deleteNode()` is implemented correctly. Works for all cases of a linked list, like removing first and last nodes. Detects invalid indexes and throws the asked for exception.
4. (30 pts.) `findItemAtIndex()` is implemented correctly. Correctly throws exception for invalid indexes. Is correctly returning a reference, so can actually be used for assignment.
5. (35 pts.) `deleteItemAtIndex()` is working. Correctly handles deleting the first and last items of list, and deleting case when list goes from 1 item to an empty list. Throws exception for invalid indexes as asked for.
6. (5 bonus pts.) Up to an additional 5 points of extra credit for correctly adding the mentioned overloaded `operator[]` and the `toReverseString()` recursive functions.

Program Style

Your programs must conform to the style and formatting guidelines given for this class. The following is a list of the guidelines that are required for the assignment to be submitted this week.

1. Most importantly, make sure you figure out how to set your indentation settings correctly. All programs must use 2 spaces for all indentation levels, and all indentation levels must be correctly indented. Also all tabs must be removed from files, and only 2 spaces used for indentation.
2. A function header must be present for member functions you define. You must give a short description of the function, and document all of the input parameters to the function, as well as the return value and data type of the function if it returns a value for the member functions, just like for regular functions. However, setter and getter methods do not require function headers.
3. You should have a document header for your class. The class header document should give a description of the class. Also you should document all private member variables that the class manages in the class document header.
4. Do not include any statements (such as `system("pause")` or inputting a key from the user to continue) that are meant to keep the terminal from going away. Do not include any code that is specific to a single operating system, such as the `system("pause")` which is Microsoft Windows specific.