Homework 6 -- Out: April 13, Wednesday -- DUE: April 27, Wednesday, 2:29pm

EC327 Introduction to Software Engineering – Spring 2022

Total: 100 points

Submission: **Failure to follow any of these guidelines **WILL** result in loss of points on the assignment.*

You will use Github Classroom to submit your solution. You can submit as many times as you want until the deadline without incurring in any penalty.

1. Accept the assignment "HW6" in the EC327 github classroom using this link: https://classroom.github.com/a/8l9dgHa8 and clone the repository on your own local machine.
2. Complete the problems in this assignment, saving your solution in a file named `List.h`.
3. Stage (add the files)
4. Commit your changes
5. *Push your changes to the repository.*

Late Homework Submissions:

Recall the late submission policy and the **fixed penalty of 20%** for submitting up to three days after the deadline. Also recall that we only grade your latest submission (e.g., if you submit both on time and after the deadline, only your late submission is graded).

** You may use any development environment you wish, as long as it is ANSI C++ compatible.* ***Make sure your code compiles and runs properly under the Linux environment on the PHO305/307 (or eng-grid) machines before submitting. IF YOUR CODE DOESN'T COMPILE, YOU WON'T GET ANY CREDIT!***

** Comment your code and use a clean, easily understandable coding convention. A coding style guide is at this link, but many others exist online:*
https://google.github.io/styleguide/cppguide.html#Comments (see "Comments")

**** Please run the following command in terminal BEFORE you try compiling your code (without the quotation marks): "*module load gcc*". If you run "*gcc --version*" you should see that your compiler version is 5.2.x. This will allow you to use the "--std=c++17" flag when compiling your code with g++.***

You are given a template class Node similar to the one that we saw in class, available as `List.h` in the starter code. This class is designed to store data of any type depending on how the objects of Node are instantiated. Your task for this homework is to implement a *list* data structure that contains instances of Node and performs basic list operations such as adding and removing elements to the list. Since we are working with templates, your code for this assignment should all be contained in a header file called `List.h` (including both class declarations and implementations).

Q1 Abstract Class [20 points]

Create an abstract class `AbstractList`, declaring the following protected members:

- An unsigned integer `numElements` that represents the number of nodes currently contained in the List
- A pointer to `Node<T> *head`, which should point to the first element in the list. For example, if a list of integers contains the following values: <`0, 1, 2, 3, 4`>, then `head` should point to the instance of Node containing 0.

The class should also declare the following public methods:

- A default constructor that sets `numElements` as 0 and `head` pointing to NULL
- A destructor that iterates through all the Nodes in the list and deletes them
- A pure virtual method `void append(T x)`
- A pure virtual method `T remove()`
- A method `bool isEmpty()` that returns true if the list is empty and false otherwise. You must implement this method in the class `AbstractList`.

Q2 LinkedList implementation. **[60 points]**

Declare a class `List` that inherits from `AbstractList` publicly.
The class should implement the pure virtual methods inherited from List as follows:

- `append` should create a new node containing the object of type T passed as a parameter, and add this node as the new head of the list. For example, if a List of integers contains the following values: <`0, 1, 2, 3, 4`> and we invoke `append(5)`, the resulting list should contain nodes with values <`5, 0, 1, 2, 3, 4`>.
- `remove` should remove the node pointed to by head, return its value, and update the head pointer to point to the previous element. For example, if a List of integers contains the following values: <`0, 1, 2, 3, 4`>, after invoking `remove()` the List will contain <`1, 2, 3, 4`> and head will point to the Node containing 1.

The class should also implement the following methods:

- A method `void append_tail(T x)` that instantiates a node containing x and adds it to the tail of the List. For example, if a List of integers contains the following values: <`0, 1, 2, 3, 4`>, after invoking `append_tail(5)` the List will contain <`0, 1, 2, 3, 4, 5`>.
- A method `void insert(T x, int pos)` that inserts a Node containing x at position pos in the List. For example, if a List of integers contains the following values: <`0, 1, 2, 3, 4`>, after invoking `insert(5, 2)` the List will contain <`0, 1, 5, 2, 3, 4`>. Note that the indexes in the List start with 0. If `pos` is invalid, the function should not add anything.
- A method `T remove_at(int pos)` that deletes the Node at position pos and returns its value. For example, if a List of integers contains the following values: <`0, 1, 2, 3, 4`>, after invoking `remove_at(2)` the List will contain <`0, 1, 3, 4`>. Note that the indexes in the List start with 0. If `pos` is invalid, the function should not do anything

Q3 Operator Overloading [20 points]

Overload the following operators for the List class:

- The subscript [] operator – once overloaded, invoking `l[x]` (where `l` is an instance of List) should return the value of the node at index x. For example, if a List `l` of integers contains the following values: <`0, 1, 2, 3, 4`>, `l[3]` should return 3. For details on how to overload the subscript operator, see page 532 of the book.
- The output stream << operator – once overloaded, invoking `cout << l` (where `l` is an instance of List) should print out all elements in `l` separated by a space. For example, if a List `l` of integers contains the following values: <`0, 1, 2, 3, 4`>, invoking `cout << l` would print the following:
  ```
  0 1 2 3 4
  ```
  If the list is empty, the function should print "`The list is empty`". You can assume that any object contained in the List overloads the << operator, and do not have to check for errors or exceptions.

All methods that you implement for Q2-Q3 should be made public. You are provided a main function to test the functionality of your program (HW6main.cpp), but do not submit the main function together with your code.

Files to submit: `List.h`

IMPORTANT NOTE: YOUR CODE WILL GO THROUGH AN AUTOGRADER SO MAKE SURE YOUR CODE TAKES THE INPUTS AS IN THE ABOVE EXAMPLES AND ITS OUTPUT IS IN THE GIVEN FORMATS. METHOD NAMES ARE CASE SENSITIVE AND YOU WILL NOT RECEIVE CREDIT IF YOU INPUT THEM INCORRECTLY. CODE THAT DOESN'T COMPILE ON THE LAB MACHINES WILL NOT BE GIVEN ANY CREDIT.