Assignment 4

Due: October 30, 2018 (11:59pm)

Assignment

You will implement and test a revised sequence class that uses a linked list to store the items.

Purpose

Ensure that you can write a small class that uses the linked list toolkit to create and manipulate a linked list.

Before starting

Read all of Chapter 5, with particular attention to 5.3 and 5.4.

How to Turn In

Submit all source code files (`*.cpp`, `*.cxx`, `*.h`) in a collection as a zip on Blackboard.

Files

- `sequence3.h`: The header file for the new sequence class. You don't have to write much of this file. Just start with the provided version. If some of your member functions are implemented as inline functions, then you may put those implementations in this file too. You might want to compare this header file with both your previous version sequence header files, and likely use that previous work as a starting point for this code. The linked list version no longer has a `CAPACITY` constant nor a `DEFAULT_CAPACITY` constant because the items are stored on a linked list instead of an array.
- `sequence3.cpp`: The implementation file for the new sequence class. You will write all of this file, which will have the invariant of the sequence class, the implementations of all the member functions of the sequence class, and the Big O time complexities of these functions.

- `node1.h` and `node1.cpp`: Copy these files to your sub-directory. They contain the linked list toolkit from Section 5.2. You may use these files without changing them. However only the documentation for the function `list_piece` is provided. You may want to write an implementation of this function to use in writing your copy constructor and overloading your assignment operator. Implementing this function is not required. However, implementing this function may make the other code easier, particularly how to handle the copy constructor and assignment operator.

- `sequence3_test.cpp`: This is the same interactive test program that you used with the earlier sequences.

- `sequence3_exam.cpp`: A non-interactive test program that will be used to grade the correctness of your new sequence class.

- `main.cpp`: A simple main file you can use during your development to test your incomplete code.

Description

Your sequence class for this assignment will differ from the your previous sequences in the following ways:

- The sequence's items are now stored on a linked list. The head pointer of the linked list is a private member variable of the sequence class. It is also suggested that you also have a tail pointer as an additional private member variable of the sequence class. The reason for the tail pointer is explained in Section 5.4 of the textbook.

- Because you are dynamically allocation memory within your sequence class, you will need to define a copy constructor, an assignment operator, and a destructor. You need to pay special attention to the value semantics of your new sequence class: you need not only to make a copy of the linked list, but also need to place the node pointers correctly.

Start by declaring the new sequence's private member variables in `sequence2.h`. This should include the dynamic array (which is declared as a pointer to a `value_type`). You will also need two `size_type` variables to keep track of the number of items in the sequence and the total size of the dynamic array. One final `size_type` will allow you to keep track of which item is the current item. After you've declared your member variables, write an invariant for the top of `sequence2.cpp`.

Start by declaring the new sequence's private member variables in `sequence3.h`. You might try declaring these variables yourself, and then compare your solution with the suggestion in Section 5.4.

Once again, do your work in small pieces. Start by writing mostly empty functions (if a function needs to return a number, start by having it return 0

and do nothing else, etc.) so that you can first get the code to compile, then add to the functions from there.

Use the interactive test program and the debugger to track down errors in your implementation. If you have an error, do not start making changes until you have identified the cause of the error.

Write your own bit of code (probably in the `main.cpp`) to declare a couple `sequence` objects and test each function by writing your own usage of them. Do this *before* you run the much more complex exam code, as this will help you figure out what works in a very simple use case. This is not required, but will probably make things easier.

When designing each function, you may find it helpful to draw out the linked list and how the pointers change on a sheet of paper before trying to write the code. Additionally, it may help to figure out the invariant *before* you begin to code.

Remember that sequence may be empty, and then many pointers will point to the `nullptr` (or `NULL`). The `precursor` and `cursor` may be the `nullptr` even when there are items in the linked list.

When a member functions needs to increase the size of the dynamic array, it is a good idea to increase that size by at least 10% (rather than by just one item).

Time analysis

Please give the Big-O of each function in your implementation, and compare them with the corresponding functions of the sequence using a dynamic array. Use the number of items in a sequence as the input size n of the algorithm. **You need to write the Big-O time analysis of each function you write**.

Optional (for extra 5% points)

Add the operators + and +=: For + operator, `x + y` contains all the items of `x`, followed by all the items in `y`. The statement `x += y` appends all the items in `y` to the end of what's already in `x`.