CS 11 Data Structures and Algorithms

Assignment 9: Linked Lists 1

Skip to Main Content

Assignment 9.1 [15 points]

This assignment will not be graded for style issues. If your code works correctly, you'll get 15 points. Style will be graded next week.

No documentation (commenting) is required for this assignment.

This assignment is based on an assignment from "Data Structures and Other Objects Using C++" by Michael Main and Walter Savitch.

Use linked lists to implement a ***sequence*** class.

Specification

The specification of the class is below. There are 9 member functions (not counting the big 3) and 2 types to define. The idea behind this class is that there will be an ***internal*** iterator, i.e., an iterator that the client cannot access, but that the class itself manages. For example, if we have a sequence object named ***s***, we would use ***s.start()*** to set the iterator to the beginning of the list, and ***s.advance()*** to move the iterator to the next node in the list.

(I'm making the analogy to iterators as a way to help you understand the point of what we are doing. If trying to think in terms of iterators is confusing, just forget about iterators and focus on following the specification below.)

What Is Due This Week

This is the first part of a two part assignment. In the next assignment you will be adding additional member functions to the class that you create in this assignment. I think this first part of the assignment is easier than the second part, so you may want to try to get an early start on next week's assignment. For this week, you are required to implement all of these functions except for attach() and remove_current(). **You are also not required to implement the big-3.**

```
typedef ____ value_type;

typedef ____ size_type;


sequence();
        // postcondition: The sequence has been initialized to an empty sequence.

void start();
        // postcondition: The first item in the sequence becomes the current item (but if the
        // sequence is empty, then there is no current item).

void advance();
        // precondition: is_item() returns true
        // Postcondition: If the current item was already the last item in the sequence, then there
        // is no longer any current item. Otherwise, the new current item is the item immediately after
        // the original current item.

void insert(const value_type& entry);
        // Postcondition: A new copy of entry has been inserted in the sequence before the
        // current item. If there was no current item, then the new entry has been inserted at the
        // front. In either case, the new item is now the current item of the sequence.

void attach(const value_type& entry);
        // Not required this week
        // Postcondition: A new copy of entry has been inserted in the sequence after the current
        // item. If there was no current item, then the new entry has been attached to the end of
        // the sequence. In either case, the new item is now the current item of the sequence.

void remove_current();
        // Not required this week
        // Precondition: is_item returns true.
        // Postcondition: The current item has been removed from the sequence, and the
        // item after this (if there is one) is now the new current item.

size_type size() const;
        // Postcondition: Returns the number of items in the sequence.

bool is_item() const;
        // Postcondition: A true return value indicates that there is a valid "current" item that
        // may be retrieved by the current member function (listed below). A false return value
        // indicates that there is no valid current item.

value_type current() const;
        // Precondition: is_item() returns true
        // Postcondition: The current item in the sequence is returned.
```

Also, your class should be placed in a namespace "cs_sequence".

Here is an example of a simple client program, to give you an idea about how sequences might be used. You can also use this to test your class, but it's far from exhaustive. Keep in mind as you write your class that you should test each member function after you write it.

```
int main() {
        sequence s;
        for (int i = 0; i < 6; i++) {
                s.insert(i);
        }

        for (s.start(); s.is_item(); s.advance()) {
```

```
                cout << s.current() << " ";
        }

        cout << endl;
}
```

This client program will print the numbers 5 4 3 2 1 0. (They are backward because insert() inserts each new item at the front of the list.)

Implementation

We will be using the following implementation of type ***sequence***, which includes 5 data members.

- numItems. Stores the number of items in the sequence.
- headPtr and tailPtr. The head and tail pointers of the linked list. If the sequence has no items, then these pointers are both NULL. The reason for the tail pointer is the attach function. Normally this function adds a new item immediately after the current node. But if there is no current node, then attach places its new item at the tail of the list, so it makes sense to keep a tail pointer around.
- cursor. Points to the node with the current item (or NULL if there is no current item).
- precursor. Points to the node before current item, or NULL if there is no current item or if the current item is the first node. Note that **precursor points to NULL if there is no current item, or, to put it another way, points to NULL whenever cursor points to NULL.** Can you figure out why we propose a precursor? The answer is the insert function, which normally adds a new item immediately before the current node. But the linked-list functions have no way of inserting a new node before a speci- fied node. We can only add new nodes after a specified node. Therefore, the insert function will work by adding the new item after the precursor node -- which is also just before the cursor node.

You must use this implementation. That means that you will have 5 data members, you won't have a header node, and **precursor must be NULL if cursor is NULL**. (What is a header node? If you don't know what it is, you probably aren't using it and shouldn't worry about it. It's not the same thing as a headPtr data member, which you WILL be using in this assignment.)

You should implement nodes using the same technique shown in the lecture videos and in the examples in the written lesson. In other words, there should not be a node class; rather, a node struct should be defined inside the sequence class.

Many students make the mistake of thinking that this assignment will very closely track with the examples done in lecture. In this assignment perhaps more than any other so far I am expecting you to use the **concepts** taught in the lesson and text to implement a class that is very different from the ones done in the lessons and the text (instead of implementing a class that is pretty similar to the ones done in the lesson and text).

I am providing my solution to the insert() function. This should make it easier to get started: you can use this insert() function and do all of this week's functions (default constructor, size(), start(), advance(), current(), and is_item()) and then write a client program to test what you have so far.

```
void sequence::insert(const value_type& entry) {
    numitems++;
    node* tempptr = new node;
    tempptr -> data = entry;

    if (headptr == NULL) {                                  // inserting into empty list.
        tempptr -> next = NULL;                             // precursor remains NULL.
        headptr = tempptr;
        tailptr = tempptr;
    } else if (cursor == NULL || cursor == headptr) {       // inserting at front.
                                                            // tailptr and precursor don't change.
        tempptr -> next = headptr;
        headptr = tempptr;
    } else {                                                // inserting anywhere else.
        tempptr -> next = cursor;                           // tailptr, headptr and precursor don't change.
        precursor -> next = tempptr;
    }

    cursor = tempptr;
}
```

Submit Your Work

Name your source code files sequence.cpp and sequence.h. No need to paste your output. Use the Assignment Submission link to submit the two files. When you submit your assignment there will be a text field in which you can add a note to me (called a "comment", but don't confuse it with a C++ comment). In this "comments" section of the submission page let me know whether the class works as required.