CS 11 Data Structures and Algorithms

Assignment 9: STL and Iterators

Return to Course Homepage

Assignment 9.1

```
#ifndef SEQUENCE_H
#define SEQUENCE_H
#include <iostream>


namespace cs_sequence {

    class sequence {
    public:
        typedef std::size_t size_type;
        typedef int value_type;
        sequence();

        void start();
        void advance();
        void insert(const value_type& entry);
        size_type size() const;
        bool is_item() const;
        value_type current() const;

    private:
        struct node {
            value_type data;
            node* next;
        };
        node* headptr;
        node* tailptr;
        node* cursor;
        node* precursor;
        size_type numitems;
    };

#include <cassert>

    sequence::sequence()
    {
        numitems = 0;
        headptr = nullptr;
        tailptr = nullptr;
        cursor = nullptr;
        precursor = nullptr;
    }






    void sequence::start() {
        cursor = headptr;
        precursor = nullptr;
    }






    void sequence::advance() {
        assert(is_item());
        precursor = cursor;
        cursor = cursor -> next;
        if (cursor == nullptr) {
            precursor = nullptr;
        }
    }
```

```
    void sequence::insert(const value_type& entry) {

        node* new_node = new node;
        new_node->data = entry;
        numitems++;

        if (cursor == headptr || cursor == nullptr) { // insert at front (or into empty list).
            new_node->next = headptr;                 // precursor remains nullptr.
            headptr = new_node;
            if (numitems == 1) {
                tailptr = new_node;
            }
        } else {                                       // inserting anywhere else
            new_node->next = cursor;                   // tailptr, headptr and precursor don't change.
            precursor->next = new_node;
        }

        cursor = new_node;
    }




    sequence::size_type sequence::size() const {
        return numitems;
    }




    bool sequence::is_item() const {
        return cursor != nullptr;
    }




    sequence::value_type sequence::current() const {
        assert(is_item());
        return cursor -> data;
    }
}
#endif
```