**Description of doubly linked list design**:

* Node class declared private underneath sequence class
* Each node class contains the following:
  + ItemType val;
  + Node\* next;
  + Node\* prev;
* An empty sequence will have its head node be a nullptr
* A typical sequence will be a doubly linked list (i.e. each node points to the next as well as the previous) that is not circular and contains no dummy node.

**Algorithm Pseudocode**:

**Remove function:**

Repeatedly until i >= num\_items:

If current node has correct value:

increment counter

If node is the head node:

Call delete\_head

Else

Call delete\_node

Else:

I++

curNode = nextNode

return counter

**Delete\_head function:**

Tmp = head->next

If tmp is not nullptr:

Tmp->previous = nullptr

Delete head

Head = tmp

Num\_items—

**Interleave function**

Repeatedly until the end of list1 or list2:

If(state = true):

State = false

Place next value of list1

Else:

State = true

Place next value of list2

Repeatedly:

Place remaining values of list1

Repeatedly:

Place remaining values of list2

Return the new list

**List of test cases:**

#include "Sequence.h"

#include <iostream>

#include <cassert>

using namespace std;

void test()

{

    // Test insertion and deletion

    Sequence s;

    assert(s.empty());

    s.insert(0, 10);

    s.insert(0, 20);

    s.insert(5);

    assert(!s.empty());

    assert(s.size() == 3);

    ItemType x = 999;

    assert(s.get(0, x) && x == 5);

    assert(s.get(1, x) && x == 20);

    assert(!s.erase(3));

    assert(s.erase(0));

    assert(s.get(0, x) && x == 20);

    assert(s.get(1, x) && x == 10);

    s.insert(20);

    assert(s.remove(20) == 2);

    assert(s.size() == 1);

    assert(s.find(10) == 0);

    s.set(0, 2);

    assert(s.find(2) == 0);

    s.dump(); // dump

    assert(s.size() == 1);

// Test swap function

    Sequence d;

    d.insert(5);

    d.insert(15);

    d.swap(s);

    assert(s.size() == 2);

    assert(s.find(15) == 1);

    assert(d.size() == 1);

    assert(d.find(2) == 0);

    d.dump(); // dump

    s.dump(); // dump

    // Copy constructor test

    Sequence b(s);

    b.insert(7);

    b.set(0,69);

    assert(b.size() == 3);

    assert(s.size() == 2);

    b.dump();

    s.dump();

    // Assignment operator test

    d = b;

    b = b;

    d.erase(0);

    assert(b.size() == 3);

    assert(d.size() == 2);

    d.dump();

    b.dump();

    // Remove and delete test

    Sequence a;

    a.insert(10);

    for(int i = 0; i < 20; i++)

    {

        a.insert(20);

    }

    assert(a.remove(20) == 20);

    assert(a.size() == 1);

    assert(a.remove(10) == 1);

    assert(a.empty());

    for(int i = 0; i < 6; i++)

    {

        a.insert(20);

    }

    assert(a.erase(5));

    assert(a.size() == 5);

    a.dump();

    // Test subsequence

    Sequence z;

    Sequence y;

    Sequence t;

    Sequence empty;

    assert(z.insert(0,30) == 0);

    assert(z.insert(1,21) == 1);

    assert(z.insert(2,63) == 2);

    assert(z.insert(3,42) == 3);

    assert(z.insert(4,17) == 4);

    assert(z.insert(5,63) == 5);

    assert(z.insert(6,17) == 6);

    assert(z.insert(7,29) == 7);

    assert(z.insert(8, 8) == 8);

    assert(z.insert(9,32) == 9);

    assert(y.insert(0,63) == 0);

    assert(y.insert(1,17) == 1);

    assert(y.insert(2,29) == 2);

    assert(t.insert(0,63) == 0);

    assert(t.insert(1,17) == 1);

    assert(t.insert(2,29) == 2);

    assert(t.insert(3,420) == 3);

    assert(subsequence(z,y) == 5);

    assert(subsequence(z,z) == 0);

    assert(subsequence(empty,y) == -1);

    assert(subsequence(y,empty) == -1);

    assert(subsequence(t,y) == 0);

    assert(subsequence(z,t) == -1);

    // Test interleave

    z.dump();

    y.dump();

    interleave(z,y,empty);

    empty.dump();

    interleave(y,z,empty);

    empty.dump();

    interleave(z,y,z);

    z.dump();

    Sequence empty2;

    Sequence empty3;

    interleave(y, empty2, empty2);

    interleave(empty3, y, empty3);

    empty2.dump();

    empty3.dump();

    assert(empty3.size() == 3);

}

int main()

{

    test();

    cout << "Passed all tests" << endl;

}