**CS 32 Project 2**

**Description of Design**

My doubly-linked list was implemented with the idea that I wanted the majority of the work to be done by the list itself. Therefore when designing the member functions I carefully thought about all the member functions I would need in order to make the implementation of the Sequence functions simple, and then implemented all those member functions. The list itself is not circular nor has dummy nodes, simply a head and tail pointer and each node contains value, a pointer to next, and a pointer to previous . In an empty list the head and tail pointers are set to nullptr. In a typical list, head points at the first node, tail points at the last node, and each node has previous and next pointers with the exception of the first node where the previous pointer is set to nullptr and the last node where the next pointer is set to nullptr.

**Pseudocode**

***int Sequence::doublyLinkedList::addToPos***

if sequence is empty

add to front

if position is 0

add to front

otherwise

increment through sequence till we reach the pos where we want to insert

if this pos is the last position

add to rear

create a new node

assign the pointers of this node to the correct objects

assigned the pointers of the object before and after this node back to this node

***int Sequence::doublyLinkedList::deleteItem***

if the sequence is empty

return

if we are deleting first item

if there is only one item

delete the item and set head/tail to nullptr

delete the item and set the respective pointers to the correct object

increment till we reach the node right before the node we want to delete

if the node is the last node

delete the item and set the respective pointers to the correct object

delete the item and set the respective pointers to the correct object

***int Sequence::insert***

if the sequence is empty

insert the value in the front

set position to zero

increment position till we find the correct position for insertion

insert value at said position

***int Sequence::remove***

increment through the entire sequence

if the value at the current position equals the desired value

erase the node at said position

return the number of times erase was called

***int Sequence::find***

return the value from the function findItem

***void Sequence::swap***

switch the sizes

switch the head pointers

switch the tail pointers

***Sequence& Sequence::operator=***

if you are not assigning the same sequence to itself

erase all values from the target sequence

copy all values from the source sequence to the target sequence

return a pointer to the target sequence

***int subsequence***

get the first value of each sequence

increment through the values of seq1 till you find one that matches the first value of seq2

if you found no values, break and return -1

increment through each value of seq1 and seq2 from their current respective positions

if at any point they are not equal, break and return -1

if we have successfully incremented through all of seq2

return the starting position of seq1 when this incrementing started

***void interleave***

if seq1 and seq2 are empty

clear result

if seq1 is empty

set result equal to seq2

if seq2 is empty

set result equal to seq1

create two temp seq from seq1 and seq2

clear result

while we have not incremented through all of result

if we have not incremented through all of seq1

insert value from seq1 and increment forward one

increment through result one

if we have not incremented through all of seq2

insert value from seq2 and increment forward one

increment through result one

**Test Cases**

Note - All tests were performed with ItemType set to string

*// Default Constructor*

void defaultConstruct()

{

Sequence a;

assert(a.empty()); // check empty

assert(a.size() == 0); // check size

assert(! a.erase(0)); // nothing to erase

assert(a.remove("paratha") == 0); // nothing to remove

ItemType x;

assert(! a.get(0, x)); // nothing to get

assert(! a.set(0, x)); // nothing to set

assert(a.find("a") == -1); // nothing to find

Sequence aa[10]; // construct array of sequences

for (int i = 0; i < 10; i++) // check all objects within array

{

assert(aa[i].empty()); // check empty

assert(aa[i].size() == 0); // check size

assert(! aa[i].erase(0)); // nothing to erase

assert(aa[i].remove("paratha") == 0); // nothing to remove

}

}

*// Insert in a specified positon*

void insertAtPos()

{

Sequence a;

for (int i = 0; i < 100; i++)

assert(a.insert(a.size(), "a"));// check insertions

assert(a.size() == 100); // check size

assert(! a.insert(-1, "a")); // check negative numbers

assert(! a.insert(101, "a")); // check numbers larger than size

for (int i = 1; i < 11; i++)

{

assert(a.insert(0, "a")); // check insertions in beginning

assert(a.size() == 100 + i); // check size

}

for (int i = 1; i < 11; i++)

{

assert(a.insert(a.size()/2, "a")); // check insertions in middle

assert(a.size() == 110 + i); // check size

}

assert(a.insert(0, "a")); // check different values inserted in same place

assert(a.insert(0, "b"));

assert(a.insert(0, "c"));

assert(a.insert(0, "d"));

assert(a.insert(0, "e"));

assert(a.size() == 125); // check size

assert(a.insert(0, "a")); // check same value inserted in different place

assert(a.insert(1, "a"));

assert(a.insert(2, "a"));

assert(a.insert(3, "a"));

assert(a.insert(4, "a"));

assert(a.size() == 130); // check size

}

*// General Insert*

void generalInsert()

{

Sequence a;

for (int i = 0; i < 10; i++)

assert(a.insert("a") == 0); // check insertions

assert(a.size() == 10); // check size

for (int i = 1; i < 11; i++)

{

assert(a.insert("a") == 0); // check insertions are going in correct pos

assert(a.size() == 10 + i); // check size

}

for (int i = 1; i < 11; i++)

{

assert(a.insert("b") == 20); // check insertions are going in correct pos

assert(a.size() == 20 + i); // check size

}

assert(a.size() == 30); // final size check

Sequence aa;

assert(aa.insert(0, "b")); // set up for checks

assert(aa.insert(1, "d"));

assert(aa.insert(2, "e"));

assert(aa.insert("a") == 0); // check value are correctly inserted in beginning

assert(aa.insert("c") == 2); // check value are correctly inserted in middle

assert(aa.insert("f") == 5); // check value are correctly inserted in end

assert(aa.size() == 6); // final size check

}

*// Erase*

void erase()

{

Sequence a;

for (int i = 0; i < 10; i++)

assert(a.insert("a") == 0); // insert values for testing

assert(a.size() == 10);

assert(a.erase(0)); // check erasing first value

assert(a.size() == 9); // check size

assert(a.erase(5)); // check erasing middle value

assert(a.size() == 8); // check size

assert(a.erase(7)); // check erasing last value

assert(a.size() == 7); // check size

assert(! a.erase(-1)); // check negative pos

assert(! a.erase(7)); // check pos larger than size()

for (int i = 1; i < 8; i++)

{

assert(a.erase(0)); // check erasing all values

assert(a.size() == 7 - i); // check size is updated each time

}

assert(a.size() == 0); // final size check

Sequence aa;

for (int i = 0; i < 100; i++) // fill a list with 100 values

assert(aa.insert(aa.size(), "a"));

assert(aa.size() == 100); // check size

for (int i = 0; i < 100; i++)

assert(aa.erase(0)); // check if whole list is erased

assert(aa.size() == 0); // final size check

}

*// Remove*

void remove()

{

Sequence a;

assert(a.insert("a") == 0); // insert values for testing

assert(a.remove("a") == 1); // check removal

assert(a.size() == 0); // check size

for (int i = 0; i < 10; i++)

assert(a.insert("a") == 0); // insert values for testing

for (int i = 0; i < 10; i++)

assert(a.insert("b") == 10); // insert values for testing

for (int i = 0; i < 10; i++)

assert(a.insert("c") == 20); // insert values for testing

for (int i = 0; i < 10; i++)

assert(a.insert("d") == 30); // insert values for testing

assert(a.size() == 40);

assert(a.remove("c") == 10); // check removal of middle values

assert(a.size() == 30); // check size

assert(a.remove("a") == 10); // check removal of beginning values

assert(a.size() == 20); // check size

assert(a.remove("d") == 10); // check removal of end values

assert(a.size() == 10); // check size

assert(a.remove("b") == 10); // check removal of all values

assert(a.size() == 0); // check size

for (int i = 0; i < 10; i++)

assert(a.insert("a") == 0); // insert values for testing

assert(a.remove("b") == 0); // check removal of nonexistent value

}

*// Get*

void get()

{

Sequence a;

ItemType x;

for (int i = 0; i < 10; i++)

assert(a.insert("a") == 0); // insert values for testing

for (int i = 0; i < 10; i++)

assert(a.get(i, x) && x == "a");// Check retrieval across all pos

assert(! a.get(-1, x)); // Check negative pos

assert(! a.get(a.size(), x)); // Check pos larger than size() - 1

Sequence aa;

assert(aa.insert(0, "b")); // insert values

assert(aa.insert(1, "d"));

assert(aa.insert(2, "e"));

assert(aa.insert("a") == 0);

assert(aa.insert("c") == 2);

assert(aa.insert("f") == 5);

assert(aa.get(0, x) && x == "a"); // check retrieval for different values

assert(aa.get(1, x) && x == "b");

assert(aa.get(2, x) && x == "c");

assert(aa.get(3, x) && x == "d");

assert(aa.get(4, x) && x == "e");

assert(aa.get(5, x) && x == "f");

}

*// Set*

void set()

{

Sequence a;

ItemType x;

for (int i = 0; i < 10; i++)

assert(a.insert("b") == 0); // insert values

for (int i = 0; i < 10; i++)

assert(a.set(i, "a")); // set values

for (int i = 0; i < 10; i++)

assert(a.get(i, x) && x == "a");// check whether values were changed

assert(! a.set(-1, "a")); // check negative pos

assert(! a.set(a.size(), "a")); // check pos larger than size() - 1

Sequence aa;

assert(aa.insert(0, "b")); // insert values

assert(aa.insert(1, "d"));

assert(aa.insert(2, "e"));

assert(aa.insert("a") == 0);

assert(aa.insert("c") == 2);

assert(aa.insert("f") == 5);

assert(aa.set(0, "z") && aa.get(0, x) && x == "z"); // check whether set work for

assert(aa.set(1, "y") && aa.get(1, x) && x == "y"); // different values across all

assert(aa.set(2, "x") && aa.get(2, x) && x == "x"); // possible pos

assert(aa.set(3, "w") && aa.get(3, x) && x == "w");

assert(aa.set(4, "s") && aa.get(4, x) && x == "s");

assert(aa.set(5, "t") && aa.get(5, x) && x == "t");

}

*// Find*

void find()

{

Sequence a;

for (int i = 0; i < 10; i++)

assert(a.insert("a") == 0); // insert values

assert(a.find("a") == 0); // check find

assert(a.insert("b") == 10); // insert value

assert(a.find("b") == 10); // check find if value is last digit

assert(a.insert(5, "b")); // insert value

assert(a.find("b") == 5); // check find if value is in middle

assert(a.insert(0, "b")); // insert value

assert(a.find("b") == 0); // check find if value is first digit

Sequence aa;

assert(aa.insert(0, "b")); // insert values

assert(aa.insert(1, "d"));

assert(aa.insert(2, "e"));

assert(aa.insert("a") == 0);

assert(aa.insert("c") == 2);

assert(aa.insert("f") == 5);

// check find for a variety of different values in different pos

assert(aa.find("a") == 0 && aa.find("b") == 1 && aa.find("c") == 2 && aa.find("d") == 3 && aa.find("e") == 4 && aa.find("f") == 5);

}

*// Swap*

void swap()

{

// Swap

Sequence s1;

Sequence s2;

ItemType x;

for (int i = 0; i < 4; i++) // insert values

s1.insert("a");

for (int i = 0; i < 8; i++) // insert values

s2.insert("b");

s1.swap(s2); // swap

assert(s1.size() == 8 && s2.size() == 4);

assert(s1.get(0, x) && x == "b"); // check values

assert(s1.get(1, x) && x == "b");

assert(s1.get(2, x) && x == "b");

assert(s1.get(3, x) && x == "b");

assert(s1.get(4, x) && x == "b");

assert(s1.get(5, x) && x == "b");

assert(s1.get(6, x) && x == "b");

assert(s1.get(7, x) && x == "b");

assert(s2.get(0, x) && x == "a");

assert(s2.get(1, x) && x == "a");

assert(s2.get(2, x) && x == "a");

assert(s2.get(3, x) && x == "a");

Sequence a;

for (int i = 0; i < 10; i++)

assert(a.insert("a") == 0); // insert values

Sequence aa;

aa.swap(a); // check swap with empty sequence

assert(a.size() == 0 && aa.size() == 10);

for (int i = 0; i < 10; i++)

assert(aa.get(i, x) && x == "a");

}

*// Destructor*

Run all these test cases using g32 on the linux server and if it reports no memory leaks, then

your destructor is most likely working correctly

*// Copy Constructor*

void copyConstructor()

{

Sequence a;

for (int i = 0; i < 10; i++)

assert(a.insert("a") == 0); // insert values

Sequence b = a; // copy construct

assert(b.size() == 10); // check size

ItemType x;

for (int i = 0; i < 10; i++)

assert(b.get(i, x) && x == "a");// check values

Sequence aa;

assert(aa.insert(0, "b")); // insert values

assert(aa.insert(1, "d"));

assert(aa.insert(2, "e"));

assert(aa.insert("a") == 0);

assert(aa.insert("c") == 2);

assert(aa.insert("f") == 5);

Sequence aaa = aa; // copy construct

// check that all different values were copied correctly into their different pos

assert(aaa.find("a") == 0 && aaa.find("b") == 1 && aaa.find("c") == 2 && aaa.find("d") == 3 && aaa.find("e") == 4 && aaa.find("f") == 5);

assert(aaa.size() == 6); // check size

}

*// Assignment Operator*

void assignmentOperator()

{

Sequence a;

for (int i = 0; i < 10; i++)

assert(a.insert("a") == 0); // insert values

Sequence b; // declare

b = a; // assignment operator

assert(b.size() == 10); // check size

ItemType x;

for (int i = 0; i < 10; i++)

assert(b.get(i, x) && x == "a");// check values

Sequence aa;

assert(aa.insert(0, "b")); // insert values

assert(aa.insert(1, "d"));

assert(aa.insert(2, "e"));

assert(aa.insert("a") == 0);

assert(aa.insert("c") == 2);

assert(aa.insert("f") == 5);

Sequence aaa; // declare

aaa = aa; // assignment operator

// check that all different values were copied correctly into their different pos

assert(aaa.find("a") == 0 && aaa.find("b") == 1 && aaa.find("c") == 2 && aaa.find("d") == 3 && aaa.find("e") == 4 && aaa.find("f") == 5);

assert(aaa.size() == 6); // check size

}

*// Subsequence*

void subsequence()

{

Sequence a; // insert values

a.insert(0, "a");

a.insert(1, "b");

a.insert(2, "c");

a.insert(3, "d");

Sequence b; // check if subseq is in the end

b.insert(0, "c");

b.insert(1, "d");

assert(subsequence(a, b) == 2);

Sequence c; // check if subseq is in the beginning

c.insert(0, "a");

c.insert(1, "b");

assert(subsequence(a, c) == 0);

Sequence d; // check if subseq is in the middle

d.insert(0, "b");

d.insert(1, "c");

assert(subsequence(a, d) == 1);

Sequence e; // check if subseq is not found

e.insert(0, "b");

e.insert(1, "d");

assert(subsequence(a, e) == -1);

Sequence f; // check if subseq is empty

assert(subsequence(a, f) == -1);

Sequence ff;

assert(subsequence(f, ff) == -1); // check if subseq and seq are empty

assert(subsequence(a, a) == 0); // check if subseq is the same

assert(subsequence(b, a) == -1); // check if subseq is larger

Sequence g; // check is subseq contains everything

g.insert(0, "a"); // except last value

g.insert(1, "b");

g.insert(2, "c");

g.insert(3, "e");

assert(subsequence(a, g) == -1);

Sequence h; // check is subseq contains everything

h.insert(0, "b"); // except first value

h.insert(1, "b");

h.insert(2, "c");

h.insert(3, "d");

assert(subsequence(a, h) == -1);

Sequence i; // check is subseq contains everything

i.insert(0, "a"); // except middle value

i.insert(1, "a");

i.insert(2, "c");

i.insert(3, "d");

assert(subsequence(a, h) == -1);

}

*// Interleave*

void interleave()

{

Sequence a; // insert values

a.insert(0, "a");

a.insert(1, "b");

a.insert(2, "c");

a.insert(3, "d");

Sequence b, c;

interleave(a, b, c); // check if seq2 is empty

assert(c.find("a") == 0 && c.find("b") == 1 && c.find("c") == 2 && c.find("d") == 3);

assert(a.size() == 4 && b.size() == 0 && c.size() == 4);

Sequence d;

interleave(b, a, d); // check if seq1 is empty

assert(d.find("a") == 0 && d.find("b") == 1 && d.find("c") == 2 && d.find("d") == 3);

assert(a.size() == 4 && b.size() == 0 && d.size() == 4);

Sequence e;

interleave(b, e, d); // check if both seq1 and seq2 are empty

assert(b.size() == 0 && e.size() == 0 && d.size() == 0);

interleave(b, a, a); // check if seq2 is the same as result

assert(a.find("a") == 0 && a.find("b") == 1 && a.find("c") == 2 && a.find("d") == 3);

assert(b.size() == 0 && a.size() == 4);

interleave(a, b, a); // check if seq1 is the same as result

assert(a.find("a") == 0 && a.find("b") == 1 && a.find("c") == 2 && a.find("d") == 3);

assert(b.size() == 0 && a.size() == 4);

b = a; // check if seq1 and seq2 have same size

interleave(a, b, e);

assert(a.size() == 4 && b.size() == 4 && e.size() == 8);

assert(e.find("a") == 0 && e.find("b") == 2 && e.find("c") == 4 && e.find("d") == 6);

interleave(a, b, a); // check if seq1 is the same as result

assert(a.size() == 8 && b.size() == 4);

assert(a.find("a") == 0 && a.find("b") == 2 && a.find("c") == 4 && a.find("d") == 6);

a = b;

interleave(b, a, a); // check if seq2 is the same as result

assert(a.size() == 8 && b.size() == 4);

assert(a.find("a") == 0 && a.find("b") == 2 && a.find("c") == 4 && a.find("d") == 6);

interleave(b, b, b); // check if seq1 and seq2 are the same as result

assert(b.size() == 8);

assert(b.find("a") == 0 && b.find("b") == 2 && b.find("c") == 4 && b.find("d") == 6);

b = c;

interleave(a, b, c); // check if seq1 is larger

assert(a.size() == 8 && b.size() == 4 && c.size() == 12);

assert(c.find("a") == 0 && c.find("b") == 3 && c.find("c") == 5 && c.find("d") == 7);

c = d;

interleave(b, a, c); // check if seq2 is larger

assert(a.size() == 8 && b.size() == 4 && c.size() == 12);

assert(c.find("a") == 0 && c.find("b") == 2 && c.find("c") == 4 && c.find("d") == 6);

Sequence z, y;

interleave(z, y, c); // check clearing

assert(z.size() == 0 && y.size() == 0 && c.size() == 0);

}