**CS 32 – Project 2 – Report**

1. My Sequence was implemented using a circularly linked list, with the head pointing to a sentinel node and looping to the end of the list. I used this implementation to reduce special cases that arise from using nullptrs.

head

undefined

m\_next

m\_previous

…

m\_value

m\_next

m\_previous

m\_value

m\_next

m\_previous

undefined

m\_next

m\_previous

m\_value

m\_next

m\_previous

head

0

1

m\_size -1

This represents an empty Sequence, as per my design implementation.

The picture below represents a Sequence of m\_size as per my design implementation. The numbers represent how my program ‘indexed’ the different nodes. These numbers do not actually exist.

2.) **Pseudo code for non-trivial functions:**

bool Sequence::insert(int pos, const ItemType& value)

{

if pos is bigger than size or negative, return false

else if pos is within 0 and size but not equal to size

create a pointer pointing to head->m\_next

Loop through from i = 0 to i < m\_size

if i == pos

create a newguy node

newguy gets value and is connected forward to p and backward to what was behind it,

and p is connected to newguy and what was behing p is connected to newguy

size increases and we return true

move p forward in unison with iterator i

else if pos is the size

create a pointer pointing to the end

create a newguy node

newguy gets value and is connected forward to the sentinel node and is connected backward

to what was the original end and the original end is connected forward to the newguy and

the sentinel is connected backward to the newguy

size increases and we return true

returns true

}

bool Sequence::insert(const ItemType& value)

{

if the size is 0

create a newguy node

newguy gets value and is connected forward to the sentinel node and is connected backward

to what was the original end and the original end is connected forward to the newguy and

the sentinel is connected backward to the newguy

size increases and we return true

create a temporary position variable and a pointer pointing to the sentinel node

Loop through from i = 0 to i < m\_size

move pointer forward, in unison with iterator i

if the value at position is greater than parameter value

set position to i and break

else

set position to size

use insert(int pos, const ItemType& value) with our newly defined position

always return true

}

bool Sequence::erase(int pos)

{

if the position is valid (non negative and within th size)

create a pointer pointing to head (p)

Loop through from i = 0 to i < m\_size

move pointer forward, in unison with iterator i

if i == pos

create a deleter pointer that points to the current value (given by p)

connect what lies behind p to whats infront of p

connect what lies in fron of p to whats behind p

delete the deleter, decrease the size and return true

return false

}

int Sequence::remove(const ItemType& value)

{

if the position is valid (non negative and within th size)

create a pointer pointing to head (p), and a removed counter

Loop through from i = 0 to i < m\_size

move pointer forward, in unison with iterator i

if i == pos

create a deleter pointer that points to the current value (given by p)

connect what lies behind p to whats infront of p

connect what lies in fron of p to whats behind p

delete the deleter, decrease the size and increment the removed counter

return removed counter

}

void Sequence::swap(Sequence& other)

{

create a temporary head pointer pointing to the others head

make others head point to head

make head point to the temporary head

make a temporary size equal to others size

make others size equal to m\_size

make m\_size equal to the temporary size

}

Sequence::~Sequence()

{

while the sentinels previous pointer doesnt point to itself (i.e dont delete the sentinel)

create a deleter pointing to the end

connect whats behind the current end to the sentinel

connect the sentinel with whats behind the current end

delete deleter pointer

delete the head

}

Sequence::Sequence(const Sequence& other)

{

set m\_size to others size

create a new head pointer and make it point to itself, making it the sentinel

create a pointer (p) pointing to others head and loop through while its not equal to head

create a new adder node

adder gets p's value

connect adder forward to the sentinel

connect adder backward to what initially lay there

connect what initally lay there forward to adder

connect the sentinel backward to adder

move p forward

}

int subsequence(const Sequence& seq1, const Sequence& seq2)

{

if seq2 is empty or bigger than seq1 then return -1

create local variables to store values from the sequences, the startposition and a boolean whether it is a sequence or not

Loop through seq1

store values at from respective positions into our two local variables using the get method

if the first element in seq2 is equal to the ith element in seq1

startpoint becomes i, and we loop through seq2 while simultaneously looping through seq1

store values at from respective positions into our two local variables using the get method

if these variables aren't equal

it isnt a subsequence, set isSequence to false and break

if after looping, there wasnt a break, isSequence is still true

return startpoint

return -1

}

void interleave(const Sequence& seq1, const Sequence& seq2, Sequence& result)

{

create a temporary sequence using a copy constructor with result (temp\_result)

if temp\_result isnt empty

erase everything currently in it

if either seq1 or seq2 is empty

make temp\_result equal to the non empty sequence

loop through both sequences until both end

if our iterator for seq1 is less than seq1 size

store the value at that position in temp\_result

increment the iterator for seq1 and temp\_result

if our iterator for seq2 is less than seq2 size

store the value at that position in temp\_result

increment the iterator for seq2 and temp\_result

Make result equal to temp\_result

}

3.) **Test cases:** (Tests were performed on a sequence of unsigned long, i.e ItemType typedef specified unsigned long)

Sequence s;

Sequence a;

assert(s.empty()); // check to see if s set is empty

assert(a.empty()); // check to see if a set is empty

for (int i = 0; i < 200; i++)

{

s.insert(i); // Insert 200 items in s

}

assert(s.size() == 200); // Check to see if s contains 200

s.swap(a); // Swap the random values from a for numbers 0-200 into a

assert(a.size() == 200); // Check to see if the size is 200

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

{

a.remove(k); //remove the first 100 numbers

}

assert(a.size() == 100); // Checks if size is now 100

Sequence r;

r.insert(r.size(), 21); // Inserts at the last position

r.insert(r.size(), 42); // Inserts at the last position

assert(r.find(42) == 1); // checks to see if find is input in the correct location

assert(!r.insert(5, 69)); // makes sure that we cant insert something into a position thats greater than the size

assert(!r.erase(3)); // makes sure we cant erase something at a position that is greater than the size

assert(r.erase(1)); // erases whats at position 1, which in this case is 42

assert(r.size() == 1);

unsigned long swag = 33;

assert(r.get(0, swag) && swag == 21); // tests to make sure that at position 0 we have 21 and x will get that value

unsigned long remix = 968;

assert(r.set(0, remix) && r.find(968) == 0); // replaces 21 with 968 at position 0 and makes sure its located there using find

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

Sequence seq1;

seq1.insert(0,30); // The following inserts test if the data is correctly inserted in the sequence

seq1.insert(1,21);

seq1.insert(2,63);

seq1.insert(3,42);

seq1.insert(4,17);

seq1.insert(5,63);

seq1.insert(6,17);

seq1.insert(7,29);

seq1.insert(8,8);

seq1.insert(9,32);

Sequence seq2;

seq2.insert(0, 17);

seq2.insert(1, 63);

seq2.insert(2, 29);

assert(subsequence(seq1, seq2)==-1); // tests that seq2 is not a subsequence of seq2

seq2 = seq1; // Tests that the assignment operator works, which in turn means the copy constructor works since it is implemented using the copy swap method

assert(subsequence(seq1, seq2) == 0); // now that seq1 and seq2 are the same, they are a subsequence of each other starting at 0

assert(subsequence(seq1, seq1) == 0); // Checks that subsequence works for aliasing.

assert(seq2.size() == seq1.size());

assert(seq2.find(42) == 3 && seq2.find(29) == 7);

Sequence seq3;

Sequence seq4;

Sequence result;

for (int p = 0; p < 55; p++) // insert 0-54 into result in positions 0-54

{

result.insert(p, p);

}

ItemType arr1[6] = {30, 21, 63, 42, 17, 63};

ItemType arr2[4] = {42,63,84,19};

for (int j = 0; j < 6; j++)

seq3.insert(j, arr1[j]);

for (int k = 0; k < 4; k++)

seq4.insert(k, arr2[k]);

interleave(seq3, seq4, result); // Interleave test(m>n)

assert(result.find(30) == 0 && result.find(19) == 7); // Makes sure that the values from seq3 and seq4 have been correctly input.

interleave(seq3, seq2, result); // Interleave test(m<n)

assert(result.find(30) == 0 && result.find(32) == 15); // Makes sure that the values from seq3 and seq2 have been correctly input.

interleave(seq2, seq1, result); //Interleave test (m = n)

assert(result.find(30) == 0 && result.size() == 2\*seq2.size()); // Makes sure the values from seq2 and seq1 have been correctly input and if so, since their sizes are equal results size should be two times seq1 or seq2's size.

interleave(seq1, seq1, seq1); // Tests for aliasing

assert(result.size() == 20);

Sequence test;

test.insert(100);

test.insert(65);

test.insert(34);

test.insert(41);

test.insert(99);

test.insert(9000);

assert(test.find(34) == 0 && test.find(9000) == test.size()-1); // Makes sure insert works correctly, making sure values are inputed in ascending order.

Sequence hi(test);

assert(hi.find(34) == 0 && test.find(9000) == test.size() - 1); // Ensures the copy constructor works properly.