**Design of doubly linked list**

My doubly linked list has a head pointer, which points to the first node, and a tail pointer, which points to the last. Each node has data of ItemType and pointer to the next and previous nodes. If there are no next or previous nodes, then the either or both pointers are set to null. If there are no nodes at all, then the head and tail pointers are set to null.

**Psuedocode**

Sequence::~Sequence

Get size

Repeatedly:

Erase the first element until none are left

Sequence::Sequence//copy constructor

Set a new pointer equal to the other head

set size to 0

set head and tail pointers to null

Repeatedly, until sequences are the same:

Insert data from the other into the new Sequence

Sequence& Sequence::operator=//overloaded assignment operator

Create a new sequence with copy constructor

Swap it with this sequence

Return this

bool Sequence::insert

If parameter is out of bounds return false

If there are no items in the list

Create new node

Set the data to the value

Set head and tail to point to new node

Increase size

Return true

If this node is the new first element in the list

Create new node

Set the data to the value

Set head to point to new node, set next to point properly

Increase size

Return true

If this node is the new last element

Create new node

Set the data to the value

Set tail to point to new node, set previous to point properly

Increase size

Return true

If this node is in the middle

Create new node

Set the data to the value

Correctly set next and previous pointers

Increase size

Return true

int Sequence::insert

look for data that is greater than given value

If there are no items in the list

Element must be the first

Create new node

Set the data to the value

Set head and tail to point to new node

Increase size

Return position

If the value is less than first element

It is the new first element

Create new node

Set the data to the value

Set head to new node, correctly set next pointer

Increase size

Return position

If the value is less than no elements

It is now the last element

Create new node

Set the data to the value

Set tail to new node, correctly set previous pointer

Increase size

Return position

If this node is in the middle

Create new node

Set the data to the value

Correctly set the pointers

Increase size

Return position

Return -1 if no operation took place

bool Sequence::erase

if the parameter is out bounds

return false

if the node is the only element

set head and tail to null

delete node

decrease size

return true

if the node is the first element

set head to point to next element

delete node

decrease size

return true

if the node is the last element

set tail to point to previous element

delete node

decrease size

return true

if the node is in the middle

set other pointers correctly

delete node

decrease size

return true

if no operation could be preformed, return false

int Sequence::remove

go through list

if data is equal to value

call erase on that node

add one to count

return count of erased items

bool Sequence::get

if the parameter is out of bounds

return false

cycle through list until desired position is reached

set the value to the data

return true

bool Sequence::set

if the parameter is out of bounds

return false

cycle through list until desired position is reached

set the data to the value

return true

int Sequence::find

cycle through list until the value is found or the end of list is reached

if end of the list is reached without finding anything

return -1

else the value was found

return the position of the value

void Sequence::swap

if this sequence is greater than the other

insert into other sequence until both have equal length

exchange the data

erase the extra data in this

if this sequence is less than the other

insert into this sequence until both have equal length

exchange the data

erase the extra data in other

if both sequences have the same length

exchange the data

int subsequence

get the sizes of both sequences

if the second sequence is empty

return -1

start at beginning of first sequence

if the beginning of the subsequence is equal to the bigger sequence

go through both sequences

if one value is not the same, break out of the loop

if the loop exits at the end of the subsequence

subsequence exists, return starting position in first sequence

if the loop exits at end of larger sequence

subsequence does not exist, return -1

void interleave

check the if result has the same size as either seq1 or seq2

check if the sequences are the same with subsequence

if seq1 and result are the same

insert seq2 into result

exit function

if seq2 and result are the same

insert seq1 into result

exit function

if result is unique

empty it of all nodes

if seq1 is empty

seq2 is the result

exit function

if seq2 is empty

seq1 is the result

exit function

set result to seq1

insert seq2 in result

**Test Values**

Sequence s;//check constructor

assert(s.empty());//test empty

assert(s.size() == 0);//test size

assert(s.insert("abba") == 0);//test int insert

assert(s.erase(0));//test erase

assert(s.empty());//test if erase worked

assert(s.find("abba") == -1);//test find, nothing to find

assert(s.insert("hey") == 0);

assert(s.remove("hey") == 1);//check remove at beginning of

sequence

for(int i = 0; i < 5; i++)

s.insert("boo");

assert(s.remove("boo") == 5);//test remove, delete everything in

a sequence

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

assert(!s.erase(10));//test erase, nothing to erase

assert(!s.erase(-10));//test erase, nothing to erase, out of

bounds

assert(!s.erase(0));//test erase

assert(s.size() == 0);//test size and if nothing happened

assert(s.insert("truth") == 0);

assert(!s.insert(-1, "compliant"));//test bool insert, out of

bounds

assert(!s.insert(3, "snap"));//test bool insert, out of bounds

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

assert(s.insert(1, "veteran"));

assert(s.insert(2, "voodoo"));

assert(s.insert(3, "chairman"));

assert(s.insert(4, "hellions"));

assert(s.insert(2, "rebellious"));

assert(s.insert("up") == 1);//test int insert, ordering

assert(s.size() == 7);

assert(s.erase(3));//test erase

assert(!s.empty());//test empty, not empty

assert(s.size() == 6);

assert(s.erase(0));//test erase, first element

assert(s.size() == 5);

assert(s.erase(4));//test erase, last element

assert(s.size() == 4);

for(int i = 0; i < 5; i++)

s.insert("overstate");

assert(s.size() == 9);

assert(s.remove("overstate") == 5);//test remove, multiple

elements

assert(s.size() == 4);

assert(s.remove("policy") == 0);//test remove, no elements

assert(s.remove("up") == 1);//test remove, one element

assert(s.size() == 3);

assert(s.remove("voodoo") == 1);//test remove, one element

assert(s.size() == 2);

assert(s.insert("demagogue") == 0);

assert(s.insert("final") == 1);

assert(s.insert("xylophone") == 4);

string x = "holy";

assert(!s.get(-1,x) && x == "holy");//test get, out of bounds

assert(!s.get(6,x) && x == "holy");//test get, out of bounds

//test get, each element of sequence

assert(s.get(0,x) && x == "demagogue");

assert(s.get(1,x) && x == "final");

assert(s.get(2,x) && x == "veteran");

assert(s.get(3,x) && x == "chairman");

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

assert(s.size() == 5);

assert(!s.empty());//test empty

x = "constitution";

assert(s.set(3, x));//test set

assert(!s.set(-1, x));//test set, out of bounds

assert(!s.set(100, x));//test set, out of bounds

assert(s.find("veteran") == 2);//test find

assert(s.find("day") == -1);//test find, element not there

Sequence s2(s);//test copy constructor

assert(s2.erase(3));

//test set

assert(s2.set(0, "voting"));

assert(s2.set(1, "polling"));

assert(s2.set(2, "supression"));

assert(s2.set(3, "jobs"));

s2.swap(s);//test swap

Sequence s3;

assert(s3.insert("jobs") == 0);//make s3, not empty, unique

s3 = s2;//check assingment operator

//check the value of s3

assert(s3.get(0,x) && x == "demagogue");

assert(s3.get(1,x) && x == "final");

assert(s3.get(2,x) && x == "veteran");

assert(s3.get(3,x) && x == "constitution");

assert(s3.get(4,x) && x == "xylophone");

assert(s3.size() == 5);

Sequence s4;

//add values into s4

assert(s4.insert(0,"saw"));

assert(s4.insert(1,"name"));

assert(s4.insert(2,"bully"));

assert(s4.insert(3,"trump"));

assert(s4.insert(4,"what"));

assert(s4.insert(5,"bully"));

assert(s4.insert(6,"what"));

assert(s4.insert(7,"could"));

assert(s4.insert(8,"like"));

assert(s4.insert(9,"years"));

Sequence s5;

//add values into s5

assert(s5.insert(0,"bully"));

assert(s5.insert(1,"what"));

assert(s5.insert(2,"could"));

assert(subsequence(s4, s5)== 5);//test subsequence

assert(subsequence(s5, s4) == -1);//test subsequnece, subsequence is larger than sequence

assert(s5.set(0,"what"));

assert(s5.set(1,"could"));

assert(subsequence(s4, s5)== -1);//test subsequence, subsequence doesn't exist

assert(s5.set(0,"could"));

assert(s5.set(1,"like"));

assert(s5.set(2,"years"));

assert(subsequence(s4, s5)== 7);//test subsequence, subsequence

starts near end

assert(s5.set(0,"like"));

assert(s5.set(1,"years"));

assert(s5.set(2,"saw"));

assert(subsequence(s4, s5)== -1);//test subsequence, subsequence starts at end

assert(s5.set(0,"years"));

assert(s5.set(1,"saw"));

assert(s5.set(2, "bully"));

assert(subsequence(s4, s5)== -1);//test subsequence, subsequence

starts at end

//create and add values to seq1

Sequence seq1;

assert(seq1.insert(0, "woman"));

assert(seq1.insert(1, "live"));

assert(seq1.insert(2, "congrats"));

assert(seq1.insert(3, "carry"));

assert(seq1.insert(4, "emergency"));

assert(seq1.insert(5, "congrats"));

//create and add values to seq2

Sequence seq2;

assert(seq2.insert(0, "carry"));

assert(seq2.insert(1, "congrats"));

assert(seq2.insert(2, "you"));

assert(seq2.insert(3, "lose"));

//create and add a value to result

Sequence result;

assert(result.insert("lalala") == 0);

interleave(seq1, seq2, result);//test interleave

//check values of result

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

assert(result.find("lalala") == -1);

assert(result.get(0,x) && x == "woman");

assert(result.get(1,x) && x == "carry");

assert(result.get(2,x) && x == "live");

assert(result.get(3,x) && x == "congrats");

assert(result.get(4,x) && x == "congrats");

assert(result.get(5,x) && x == "you");

assert(result.get(6,x) && x == "carry");

assert(result.get(7,x) && x == "lose");

assert(result.get(8,x) && x == "emergency");

assert(result.get(9,x) && x == "congrats");

interleave(seq1, seq2, seq1);//test interleave with aliasing

//check values of seq1

assert(seq1.size() == 10);

assert(seq1.get(0,x) && x == "woman");

assert(seq1.get(1,x) && x == "carry");

assert(seq1.get(2,x) && x == "live");

assert(seq1.get(3,x) && x == "congrats");

assert(seq1.get(4,x) && x == "congrats");

assert(seq1.get(5,x) && x == "you");

assert(seq1.get(6,x) && x == "carry");

assert(seq1.get(7,x) && x == "lose");

assert(seq1.get(8,x) && x == "emergency");

assert(seq1.get(9,x) && x == "congrats");