Description:

My sequence class contains following private members: a Node structure, a pointer to the head Node, and a int member m\_size that keeps track of the number of Nodes in the sequence. The link list is not circular and has no dummy Node; in each Node there are a value, a pointer points to its previous Node and one to its after. A typical sequence should have a bunch of Nodes, each of which store a value and points to the Node ahead and after, with the head pointer points to the first of these Nodes. An empty sequence just has a head pointer points to nullptr and m\_size equals to 0.

Pseudocode:

Copy constructor

{

copy size from src

check if src is empty

create a new node and store it in head

while currently not at the last node of source

copy the value from src node

point to prev node

create a new node and store it in next

advance pointers to the next node

fill the last node

}

Assignment operator

{

check aliasing

delete whatever in the sequence

check if src is empty

while currently not at the last node of source

copy the value from src node

point to prev node

create a new node and store it in next

advance pointers to the next node

fill the last node

return the address of this sequence

}

Insert value to pos

{

check if pos is in bound

add to front if pos is 0

while current pos is not pos

go to next node which is one ahead of current pos

fill the latest node

change the next link of the node ahead

change the prev link of the next node if there is one

update size

}

Insert value

{

add to front if value is less or equal than the first node

while is not at the last node

break if value is less or equal than current node

advance to next node

update pos

fill the latest node

change the next link of the node ahead

change the prev link of the next node if there is one

update size

}

Erase node at pos

{

check if pos is in bound

delete the first node if pos is zero

while current pos is not pos

go to next node which is one ahead of current pos

let deleteThis points to the node to delete

change the next link of the node ahead

change the prev link of the next node if there is one

delete the node

update size

}

Remove all the items that equals to value

{

while it does not go pass the last node and the node equals value

(this makes sure after the loop the first node is not value)

erase this node

update the pointer so it points to the start of the sequence

add one to number

while it is not the last node

if the value of next node is value

add one to number

erase next node

continue to check the current next node

advance to the next node

update pos

}

Swap the sequence with another sequence

{

exchange the head pointer of two sequences

exchange the size of two sequences

}

Add value to front

{

create a new node and fill it

change the prev link of the old first node if there is one

update head to point to the new node

update size

}

Delete first node

{

return if there is no first node

set the second node’s prev to null if there is one

update head

delete the first node

update size

}

Delete all nodes in the sequence

{

return if there is no node

while it is not the last node

update next pointer to points to next node

delete current node

advance to next node

delete the last node

}

Check if seq2 the subsequence of seq1

{

return 0 if aliasing or seq2 is empty

return -1 if seq1 is less than seq2 (not possible to contain seq2)

get the first value of seq2

find its first appearance in seq1

return -1 if the first value of seq2 is not in seq1

while there is more to check in seq1

get current values from each sequence

if two values are the same

both advance to next pos

if seq2 reaches the end

return starting point

else

if value1 equals the first value of seq2

seq2 starts a new comparison from 1

else

seq2 starts from 0

advance seq1 to next pos

return -1 for not finding anything

}

Interleave seq1 and seq2 into result

{

if either of seq1 or seq2 is empty

assign the other one to result

delete all the nodes of result

while either of the sequence is not finished

if seq1 is not finished

get value from current pos1 in seq1

insert it to posr in result

update posr and pos1

if seq2 is not finished

get value from current pos2 in seq2

insert it to posr in result

update posr and pos2

}

Test cases:

 Sequence t1;

    string a;

    assert(t1.empty() == true); // test empty func

    assert(t1.size() == 0); // test size func

    assert(t1.insert("apple") == 0); // test insert

    assert(t1.erase(0) == true); // test erase

    assert(t1.insert(3,"hahaha") == false); // test insert out of bound

    assert(t1.insert(0,"hk") == true); // test insert to pos

    assert(t1.insert("apple") == 0);

    assert(t1.insert("sss") == 2); // test insert comparison

    assert(t1.size() == 3); // test size

    assert(t1.insert(1,"ppap") == true);

    assert(t1.erase(10) == false); // test erase out bound

    assert(t1.erase(1) == true); // test successful erase

    assert(t1.get(1,a) == true && a == "hk"); // test get

    assert(t1.insert(a) == 1); test insert

    assert(t1.remove(a) == 2 && t1.size() == 2); // test remove

    assert(t1.insert(4,"blah") == false);

    assert(t1.insert("blah") == 1);

    assert(t1.insert("you") == 3); // test insert

    assert(t1.set(2,"mushroom") == true); // test set

    assert(t1.insert("apple") == 0);

    assert(t1.insert("apple") == 0);

    assert(t1.remove("apple") == 3); // test remove when the first n items all equal to value

    assert(t1.insert("apple") == 0);

    assert(t1.insert (1,"you") == true);

    assert(t1.find("you") == 1); // test find

    assert(t1.find("I") == -1); // test unsuccessful find

    //apple you blah mushroom you

    Sequence t2;

    assert(t2.insert("now") == 0);

    assert(t2.insert("now") == 0);

    assert(t2.insert("now") == 0);

    assert(t2.insert("now") == 0);

    assert(t2.insert("now") == 0);

    assert(t2.insert("now") == 0);

    assert(t2.insert("now") == 0);

    assert(t2.insert("now") == 0); // create

    //now now now now now now now now

//    t1.swap(t2); // test swap

//    assert(t1.size() == 8);

//    assert(t2.get(3,a) == true && a == "mushroom");

//    assert(t2.get(6,a) == false); // test swap

//    assert(t1.remove("now") == 8); // test remove when all items are value

    Sequence t3;

    assert(subsequence(t1, t2) == -1); // test subsequence

    assert(t2.remove("now") == 8);

    interleave(t1, t2, t3); // test interleave with empty seq

    t2.insert("blah");

    assert(subsequence(t1, t2) == 2);

    t2.insert("mushroom");

    assert(subsequence(t1, t2) == 2); // test subsequence

    t2.insert(0, "you");

    assert(subsequence(t1, t2) == 1);

    t1.insert("you");

    assert(subsequence(t1, t2) == 2); // test subsequence when the breaking node is the first node of seq2

    t1.swap(t2);

    assert(subsequence(t1, t2) == -1);

    t2.erase(0);

    t2.erase(0);

assert(subsequence(t1, t2) == -1);   //you blah mushroom    //you blah mushroom you

// test subsequence when seq1 is less than seq2

    t1.erase(0);

    interleave(t1, t2, t3); //test interleave seq1 < seq2

    t1.swap(t2);

interleave(t1, t2, t3); //test interleave seq2 < seq1

    cout << "pass all test" << endl;