This is a doubly-linked list implementation. Sequence itself has a head pointer to point at the node at the beginning of the list. Each node contains a m\_prev pointer and a m\_next pointer, to point to the previous and next node in the list respectively, as well as m\_pos to mark position in the sequence and m\_data for the values we want to keep track of.

**PSEUDOCODE**

**bool insert:**

if position is invalid, return false

if inserting front of list or if list empty

create new node

Set position = 0, set text = value, prev pointer = nullptr, next pointer to head,

while next pointer != nullptr

Repeatedly run through list to increase position of node after insert

point head pointer to new node;

    increase size of list

else if end/

Run through to find position of insertion

create new node

Set position = pos, set text = value, prev pointer = position before insertion, next pointer = nullptr, position of insertion pointer = new node pointer

increase size of list

else if middle

Run through to find position right before insertion

Make temporary pointer(temp1) to point this position

Make temporary pointer (temp2) to point to next position

create new node

Set position after temp1 to new node

Set new node’s previous to temp 1

Set new node’s previous to temp 2

Set position before temp2 to new node

Set position = pos, set text = value

Repeatedly run through to increase position of each node after

increase size of list

**int insert:**

repeatedly run through to find position that is greater than/equal to value

if found, insert value into position

if not found, insert into position #size();

**int remove**

Set counter for removed items = 0

Repeatedly search for nodes with value by

set temporary pointers (1) to position before deletion (2) position of deletion

Search for position before deletion

if found, erase and increment counter

increment pointer

Check last position if has value

If so, erase and increment counter

return counter

**bool erase**

if position is invalid, return false

if list is one element

set head of list to nullptr, delete node, decrease size

else if deleting front of list

set second node in list’s prev pointer to null

set front of list pointer to second node

repeatedly run through list to modify private position

     delete first node

decrease size

else if delete middle of list

Repeatedly run through code to find position right before deletion

Make temporary pointer(temp1) to point this position

Make temporary pointer (temp2) to point position after deletion

set temp1’s next pointer to temp2 (skipping over node-to-be-deleted)

Set temp2’s prev pointer to temp 1

delete node;

repeatedly run through to modify position after deleted node.

decrease size

else end of list

run through to last node

temporary pointer to second-to-last node

set second-to-last node’s next pointer to null

delete last node

**interleave / subsequence:**

did not implement in time.

**TEST CASES**

1)

ItemType is std::string;

Sequence s; //Default constructor

value = “Hi”;

**//Empty sequence**

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

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

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

assert(s.erase(0) == false); //nothing to erase

assert(s.erase(1) == false); //invalid position

assert(s.get(0, value)==false); //nothing to get

assert(s.set(0, value)== false); nothing to set

assert (s.find(value)== -1); nothing to find

s.insert(0, "a"); **//one node sequence**

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

assert(s.empty()==false); // test if empty

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

assert(s.get(0, value)==true); //should be able to copy into value

assert(s.erase(0) == true); //should be able to erase fine

s.insert(0, "a")

assert(s.remove("a") == 1); //should be able to remove one element properly.

s.insert(0, "a");

assert(s.get(0, value)==true); //value should be “a”

assert (s.find(value)== 1) //should be able to find a

assert(s.set(0, value)== true); should be able to set position 0 to a.

s.insert(1, "b"); **//multi-node sequence**

s.insert(2, "c");

s.insert(3, "b");

s.insert(4, "e");

assert(s.remove("b") == 2); //test if remove

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

s.insert(1, "b");

s.insert(0, "O"); //insert at beginning of sequence

assert (s.find(“O”)== 1) //find if node at beginning of sequence

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

s.insert(5, "f");//insert end of sequence

assert (s.find(“f”)== 1) //find if node at end of sequence

s.insert(3, "g"); //insert middle of sequence

assert (s.find(“g”)== 1) //find if node at middle of sequence

assert(s.remove("O") == 1)//remove node at beginning of sequence

assert(s.erase(2) == true); //erase node at middle of sequence

s.insert(s.remove(f")==1);//remove node at end of sequence

s.insert(5, “f”);

string x;

assert(s.get(0, x) && x == "a"); //testing get

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

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

assert(s.set(5, “hi”)== true; //should be able to set “hi” in place of “f”

assert(s.remove("hi") == 1);

Sequence s should look like “a b c d e”

Sequence t; **//multi-node sequence if value in node appears more than once** t.insert(0, "b");

t.insert(1, "b");

t.insert(2, "c");

t.insert(3, "b");

t.insert(4, "e");

assert(t.insert(“a”)==0); // testing int insert if value does not appear

//insertion at beginning

assert(t.insert(“f”)==6); //testing if value is larger than item in list

// (end of list);

assert(t.insert(“b”)==1;//testing if same value appears in list

// (middle of list)

assert(t.find(“b”)==1);//testing if multiple b values

Sequence t now looks like “ a b b b c b e f “

s.swap(t); //**Testing swap function/copy constructor**

assert(s.size()==8); //did m\_size swap correctly?

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

assert(s.find(“f”)==7);// “f” does not originally appear in Sequence s

//if correctly swapped, should be true

assert(t.find(“d”)==3);// “d:” does not originally appear in Sequence t

2) **Testing subsequence function**

ItemType is unsigned long

(Pretend I used insert function to insert all values in sequences)

Sequence seq1 is { 30 21 63 42 17 63 17 29 8 32}

if Sequence seq2 is {63 17 29}//subsequence found in

assert(subsequence(seq1, seq2)== 5) //middle

if Sequence seq2 is {30 21 63} //subsequence in beginning

assert(subsequence(seq1, seq2)== 0)

if Sequence seq2 is {32} //subsequence in end

assert(subsequence(seq1, seq2)== 9)//one element subsequence

if Sequence seq2 is {1 2 3} //subsequence does not appear

assert(subsequence(seq1, seq2)== -1)

if Sequence seq2 is {} //empty sequence

assert(subsequence(seq1, seq2)== -1)

3) **Testing interleave function**

ItemType is unsigned long

let m = number of values in seq1, n = number of values in seq2,

(Pretend I used insert function to insert all values in sequences)

Sequence seq1 is { 30 21 63 42 17}

if Sequence result;

if Sequence seq2 is {1 2 3} //if m > n

result is {30 1 21 2 63 3 42 17}

if Sequence seq2 is {1 2 3 4 5 6} //if m < n

result is {30 1 21 2 63 3 42 4 17 5 6}

if Sequence seq2 is {1 2 3 4 5 } //if m == n

result is {30 1 21 2 63 3 42 4 17 5 }

if Sequence seq2 is {} // if empty

result is {30 21 63 42 17}

if Sequence seq1 is {}

if Sequence seq2 is {1 2 3 4 5 } // if seq 1 is empty

result is {1 2 3 4 5 }

Also check if seq1 and result are the same

(either do nothing or make temporary sequence and copy function out)