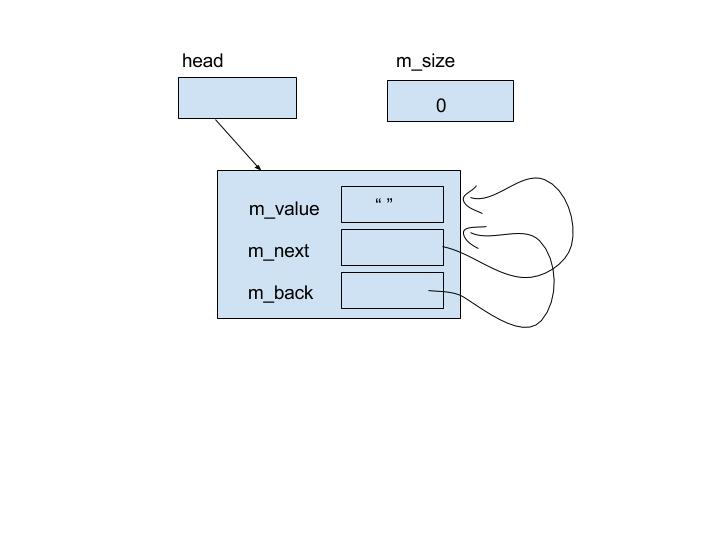
Christina Oliveira

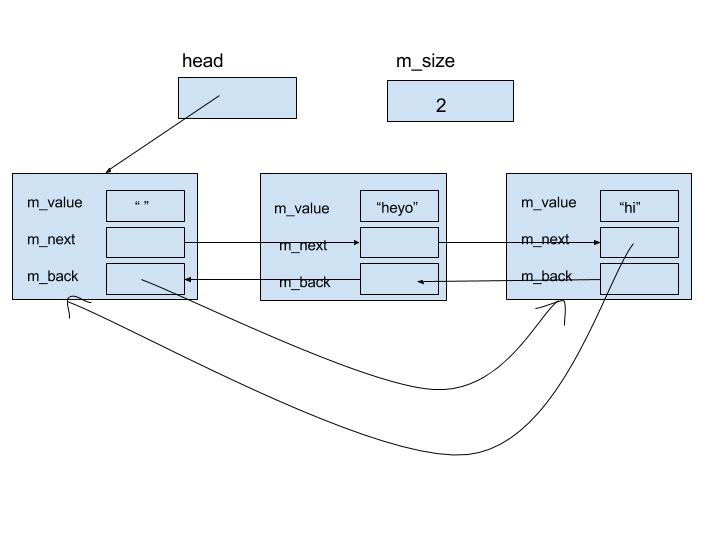
UID:204-803-448

Project 2 Report

To implement these functions, I created a circular doubly-linked list. Each Node contains a pointer to the next and previous node and space to store a value of type ItemType. I also have a counter to keep track of the number of values in my list, m\_size. An empty Sequence structured in this manner with string as the ItemType can be visualized as such:

And a typical Sequence structured the same way can be visualised like so (see next page):

Notice the head node does not contain a value (it is a dummy node).



**Pseudocode:**

**Sequence::insert**(specified position, value to be inserted){

if we’re not adding to the end of the sequence or to an empty sequence:

Traverse the list to the Node before the specified position

Insert new Node (link it up and reconnect surrounding nodes)

increment size

else

Go back one Node from the head pointer

Insert new Node (link it up and reconnect surrounding nodes)

increment size

}

**Sequence::insert**(value){

if the list is empty

Insert new Node using insert function

else

Traverse the list until the next value is greater than or equal to the value given

Insert new Node with value here using insert function

}

**Sequence::erase**(specified position to delete node at){

Traverse the list until reaching the node to be deleted

Correctly connect surrounding nodes around the one to be deleted

Delete the node

Decrement the size and return

}

**Sequence::remove(**all nodes w/ specified value){

Traverse the list

If a node has a value == the specified value

delete said node using erase function

increment count of deleted nodes

}

**Sequence::swap{**

Exchange heads using a temp Node pointer

Exchange sizes using a temp variable

}

**subsequence**{

Traverse sequence1 looking for the first item in sequence2

If found

Check, starting at that point for the rest of sequence2

if there are any non-matches

break and continue looking for first items

else if it is a full match return true

}

**interleave**{

Using a temp sequence as the sequence to be returned

Empty the temp sequence using erase function

if both sequences are empty

Return the empty sequence

if one sequence is empty

return the other sequence

else if both contain items

repeatedly:

while there are still items in seq1, add an item to the return seq

while there are still items in seq2, add an item to the return seq

return the return sequence

}

**Test Cases:**

//Tests for ItemType = unsigned long

//tests for insert(pos, value)

Sequence t;

assert(t.size()==0 && t.empty()); // constructor and basic accessors are working properly

t.insert(0,2); //insert item when seq empty

t.insert(t.size(),4); //insert item at end when seq not empty

t.insert(0, 6); //insert item at start when seq not empty

t.insert(1, 8); //insert item in middle

assert(t.find(2) == 2); //these tests ensure that the inserts went where they needed to

assert(t.find(4) == 3);

assert(t.find(6) == 0);

assert(t.find(8) == 1);

assert(!t.insert(9, 12)); //to ensure we can’t insert something at pos > size

//tests for insert(value)

t.insert(1); //insert item less than all others

t.insert(10); //insert item greater than all others

t.insert (5); //insert item in the middle

t.insert(5); //insert item with same value as another item in the middle

assert(t.find(1) == 0); //these tests ensure that the inserts went where they needed to

assert(t.find(2) == 5);

assert(t.find(4) == 6);

assert(t.find(5) == 1);

assert(t.find(6) == 3);

assert(t.find(8) == 4);

assert(t.find(10) == 7);

//t is now 1 5 5 6 8 2 4 10

assert(t.size() == 8); //to ensure the size is being incremented

//tests for erase

t.erase(0); //erase item at pos 0

t.erase(t.size()-1); //erase item at end

t.erase(1); //erase item in middle

assert(t.find(5) == 0); //these tests ensure that the deletions happened where they were supposed to

assert(t.find(6) == 1);

assert(t.find(8) == 2);

assert(t.find(2) == 3);

assert(t.find(4) == 4);

assert(!t.erase(t.size())); //cant erase item at pos out of bounds

//t is now 5 6 8 2 4

assert(t.size() == 5); //to ensure the size is being decremented

//tests for remove

t.insert(5);

assert(t.remove(5)== 2); //removing more than one item

assert(t.remove(80)== 0); //removing no items

assert(t.find(6) == 0); //these asserts assure things were deleted correctly

assert(t.find(8) == 1);

assert(t.find(2) == 2);

assert(t.find(4) == 3);

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

Sequence s;

for(int i = 0; i < 10; i++) //fill s with 5s

{

s.insert(5);

}

s.remove(5); //tests for removing all items in a sequence

assert(s.size() == 0); //size should be zero if all items were removed

//tests for swap

Sequence bruh, dude;

bruh.insert(0,4); //fill sequences

bruh.insert(1,6);

bruh.insert(2,0);

bruh.insert(3,4);

bruh.insert(4,8);

dude.insert(0,9);

dude.insert(1,9);

dude.insert(2,9);

swap(bruh, dude); //swap two sequences

assert(bruh.size()==3); //ensure sizes are swapped

assert(bruh.find(9)==0); //ensure swapping went correctly

//tests for subsequence

assert(subsequence(t,s)==-1); //seq2 is an empty sequence, so should return -1

s.insert(0,2);

s.insert(1,4); //s is now 2 4 and t is 6 8 2 4

assert(subsequence(t,s)==2); //checks for basic sequence functionality

Sequence uno, dos;

uno.insert(0,4);

uno.insert(1,6);

uno.insert(2,0);

uno.insert(3,4);

uno.insert(4,8);

assert(subsequence(uno, t)==-1); //tests for sequence partially contained

dos.insert(0,4);

dos.insert(1,8);

assert(subsequence(uno, dos)==3); //tests for partial sequence at start and full later

//tests for interleave

Sequence result;

result.insert(4); //insert value to make sure interleave will clear result

interleave(s, t, result);

assert(result.find(2)==0); //ensure values inserted correctly

assert(result.find(6)==1);

assert(result.find(4)==2);

assert(result.find(8)==3);

Sequence empty;

interleave(empty, s, result); //one sequence is empty, result should be the same as s

assert(result.find(2)==0); //ensure values inserted correctly

assert(result.find(4)==1);

assert(result.size()==2); //ensure interleave is changing results size

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

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