**CS32 Project 2 Report**

**Ben He**

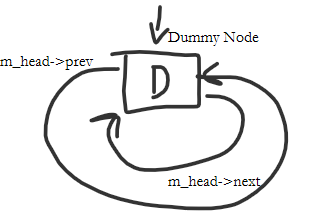
**UID: 804962948**

**Design Description**

The doubly-linked list I used in my implementation was circular with the head pointer (which I named m\_head) pointing to a dummy node, so the first element of a Sequence would actually be at m\_head->next. Also, because it’s a doubly-linked list, each Node has three variables: a pointer to the next Node in the Sequence, a pointer to the previous Node in the Sequence, and the value of this Node. In a Sequence that contains at least one Node, m\_head->prev points to the last Node of the linked list and m\_head->next points to the first Node.

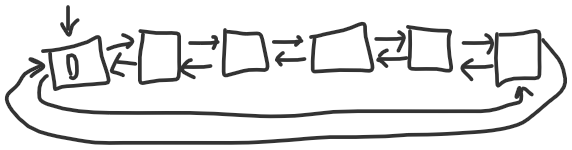
**Empty Sequence (created in constructor):**

m\_head



**Typical Sequence:**

m\_head



**Non-Trivial Algorithms**

**Sequence::Sequence()**

-set Sequence size to zero

-create dummy node

-set head pointer to dummy node

-set next and previous pointers of dummy node to itself

**Sequence::Sequence(const Sequence& other)**

-create empty Sequence the same way as Sequence()

-start from first element of other Sequence

-repeatedly until end of other:

-insert the value of other’s current element into this with insert()

**Sequence& Sequence::operator=(const Sequence& other)**

-if this and other are not aliases,

-copy construct temporary Sequence with other

-swap this Sequence with temporary Sequence with swap()

-return this Sequence

**Sequence::~Sequence()**

-start from first element of this Sequence

-repeatedly until end of Sequence:

-record location of current element

-move to location of next element

-delete current element

-delete dummy node

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

-if pos is within Sequence size,

-create new Node with value in parameter

-locate address of current Node at pos

-set new Node’s next to address of Node at pos

-set new Node’s prev to address of Node before Node at pos

-set Node before Node at pos’s next to address of new Node

-set Node at pos’s prev to address of new Node

-increment size of Sequence

-return true

-else,

-return false

**int Sequence::insert(const ItemType& value)**

-initialize integer index tracker to be 0

-start from first element of Sequence

-repeatedly until end of Sequence:

-if value from parameter is less than or equal to value of current Node,

-break

-else,

-move to next element in Sequence

-increment index tracker by 1

-insert Node with value from parameter into position recorded in index tracker

-return value of index tracker

**bool Sequence::erase(int pos)**

-if pos is within the size of Sequence,

-locate address of Node at pos

-set next of the Node before pos Node to address of Node after pos Node

-set prev of Node after pos Node to address of Node before pos Node

-delete pos Node

-decrement size of Sequence

-return true

-else,

-return false

**int Sequence::remove(const ItemType& value)**

-initialize integer number of removed elements to 0

-initialize integer Node position tracker to 0

-repeatedly, until node position tracker equals -1:

-set node position tracker to position returned by find(value)

-if node position tracker is not -1,

-erase the Node at that position

-increment number of removed elements by one

-return number of removed elements

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

-if pos is within the size of Sequence,

-locate the address of the Node at pos

-set value from parameter to value of pos Node

-return true

-else,

-return false

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

-if pos is within the size of Sequence,

-locate the address of the Node at pos

-set value of pos Node to value from parameter

-return true

-else,

-return false

**int Sequence::find(const ItemType& value) const**

-initialize integer position tracker index to 0

-start from first elements of Sequence

-repeatedly, until end of Sequence:

-if value of current Node matches value from parameter,

-break

-else,

-increment position tracker

-move to next Node

-if position tracker passed through whole Sequence without finding match,

-set position tracker to -1

-return value of position tracker

**void Sequence::swap(Sequence& other)**

-set temporary Node pointer to head of this Sequence

-set the head of this Sequence to the head of other Sequence

-set head of other Sequence to address from temporary Node pointer

-set temporary integer to size of this Sequence

-set size of this Sequence to size of other Sequence

-set size of other Sequence to value of temporary integer

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

-create two ItemType Node value tracker variables for each parameter Sequence

-initialize subsequence beginning tracker to -1

-set seq2 value tracker to seq2’s first element

-declare seq1 loop index outside of for loop

-repeatedly, starting from first element of seq1 and until last element:

-update seq1 value tracker to current seq1 Node’s value

-if seq1 value tracker matches seq2 value tracker,

-record index where first match was found

-repeatedly, starting from first element of seq2 until the last element:

-update seq1 value tracker to current seq1 Node’s value continuing from location of first match

-update seq2 value tracker to current seq2 Node’s value

-if seq1 value tracker no longer matches seq2 value tracker,

-reset seq2 value tracker back to first element of seq2

-break

-increment seq1 index to keep on track within seq1

-if inner loop completes without mismatch in corresponding Nodes of either sequence,

-set subsequence beginner tracker to index where first match was found

-return value of subsequence beginner tracker

**void interleave(const Sequence& seq1, const Sequence& ceq2, Sequence& result)**

-create empty temporary Sequence

-initialize temporary index for navigating temporary Sequence to 0

-initialize parameter index for navigating parameter Sequences to 0

-check which parameter Sequence has larger size

-repeatedly, until the end of the smaller Sequence:

-insert value of seq1’s Node at parameter index into temporary Sequence at temporary index

-increment temporary index

-insert value of seq2’s Node at parameter index into temporary Sequence at temporary index

-increment both temporary and parameter indexes

-repeatedly, until the end of the larger Sequence:

-if seq1 was the larger sequence,

-get value of seq1’s Node at parameter index

-else,

-get value of seq2’s Node at parameter index

-insert value of larger Sequence’s Node at parameter index into temporary Sequence at temporary index

-increment both temporary and parameter indexes

-swap result Sequence with temporary Sequence

**Test Cases**

**//When using ItemType = int**

Sequence s;

**//these test to make sure both forms of insert and empty functions work //properly**

assert(s.empty() == true);

assert(s.insert(0, 1) == true);

assert(s.insert(1, 2) == true);

assert(s.insert(2, 3) == true);

assert(s.insert(3, 4) == true);

**//make sure that invalid pos input will cause insert (2 parameter) to return //false**

assert(s.insert(5, 3) == false);

assert(s.insert(2) == 1);

assert(s.insert(2) == 1);

assert(s.empty() == false);

**//these test size and erase functions**

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

assert(s.erase(1) == true);

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

//assert(s.erase(4) == true);

**//these test to make sure get and erase functions work properly**

int test1;

assert(s.get(0, test1) == true && test1 == 1);

assert(s.get(1, test1) == true && test1 == 2);

assert(s.get(2, test1) == true && test1 == 2);

assert(s.get(3, test1) == true && test1 == 3);

assert(s.get(4, test1) == true && test1 == 4);

assert(s.erase(0) == true);

**//makes sure erase returns false when given invalid parameter**

assert(s.erase(5) == false);

assert(s.get(0, test1) == true && test1 == 2);

assert(s.get(1, test1) == true && test1 == 2);

assert(s.get(2, test1) == true && test1 == 3);

assert(s.get(3, test1) == true && test1 == 4);

**//test set function**

assert(s.set(2, 10) == true);

**//test to make sure set function returns false when given invalid pos input**

assert(s.set(4, 20) == false);

assert(s.get(0, test1) == true && test1 == 2);

assert(s.get(1, test1) == true && test1 == 2);

assert(s.get(2, test1) == true && test1 == 10);

assert(s.get(3, test1) == true && test1 == 4);

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

**//testing find function**

assert(s.find(2) == 0);

assert(s.find(10) == 2);

**//make sure correct return when invalid input**

assert(s.find(20) == -1);

assert(s.find(-2) == -1);

**//testing the copy constructor to make sure it is identical to passed object**

Sequence s2(s);

int test2;

assert(s2.get(0, test2) == true && test2 == 2);

assert(s2.get(1, test2) == true && test2 == 2);

assert(s2.get(2, test2) == true && test2 == 10);

assert(s2.get(3, test2) == true && test2 == 4);

assert(s2.size() == 4);

assert(s2.find(2) == 0);

assert(s2.find(10) == 2);

assert(s2.find(20) == -1);

assert(s2.find(-2) == -1);

**//testing to make sure the swap function works properly**

Sequence s6;

s6.swap(s2);

int test6;

assert(s6.get(0, test6) == true && test6 == 2);

assert(s6.get(1, test6) == true && test6 == 2);

assert(s6.get(2, test6) == true && test6 == 10);

assert(s6.get(3, test6) == true && test6 == 4);

assert(s6.size() == 4);

assert(s6.find(2) == 0);

assert(s6.find(10) == 2);

assert(s6.find(20) == -1);

assert(s6.find(-2) == -1);

**//testing remove function returns correct value and erases correctly**

assert(s6.remove(2) == 2);

assert(s6.remove(10) == 1);

assert(s6.remove(35) == 0);

assert(s6.size() == 1);

assert(s6.get(0, test6) == true && test6 == 4);

Sequence s3;

s3.insert(0, 30);

s3.insert(1, 21);

s3.insert(2, 63);

s3.insert(3, 42);

s3.insert(4, 17);

s3.insert(5, 63);

s3.insert(6, 17);

s3.insert(7, 29);

s3.insert(8, 8);

s3.insert(9, 32);

s3.insert(10, 17);

s3.insert(11, 63);

s3.insert(12, 29);

Sequence s4;

s4.insert(0, 63);

s4.insert(1, 17);

s4.insert(2, 29);

s4.insert(3, 8);

Sequence s5;

s5.insert(0, 17);

s5.insert(1, 63);

s5.insert(2, 29);

**//testing that subsequence function is working properly**

assert(subsequence(s3, s4) == 5);

assert(subsequence(s3, s5) == 10);

Sequence seq1;

seq1.insert(0, 30);

seq1.insert(1, 21);

seq1.insert(2, 63);

seq1.insert(3, 42);

seq1.insert(4, 17);

seq1.insert(5, 63);

Sequence seq2;

seq2.insert(0, 42);

seq2.insert(1, 63);

seq2.insert(2, 84);

seq2.insert(3, 19);

Sequence combSeq;

**//testing that the interleave function works properly**

interleave(seq1, seq2, combSeq);

assert(combSeq.get(0, test2) == true && test2 == 30);

assert(combSeq.get(1, test2) == true && test2 == 42);

assert(combSeq.get(2, test2) == true && test2 == 21);

assert(combSeq.get(3, test2) == true && test2 == 63);

assert(combSeq.get(4, test2) == true && test2 == 63);

assert(combSeq.get(5, test2) == true && test2 == 84);

assert(combSeq.get(6, test2) == true && test2 == 42);

assert(combSeq.get(7, test2) == true && test2 == 19);

assert(combSeq.get(8, test2) == true && test2 == 17);

assert(combSeq.get(9, test2) == true && test2 == 63);

**//making sure that interleave works even with aliasing**

interleave(seq1, seq2, seq1);

assert(seq1.get(0, test2) == true && test2 == 30);

assert(seq1.get(1, test2) == true && test2 == 42);

assert(seq1.get(2, test2) == true && test2 == 21);

assert(seq1.get(3, test2) == true && test2 == 63);

assert(seq1.get(4, test2) == true && test2 == 63);

assert(seq1.get(5, test2) == true && test2 == 84);

assert(seq1.get(6, test2) == true && test2 == 42);

assert(seq1.get(7, test2) == true && test2 == 19);

assert(seq1.get(8, test2) == true && test2 == 17);

assert(seq1.get(9, test2) == true && test2 == 63);

**//testing to make sure interleave provides an empty Sequence if both //parameters are empty**

Sequence seq3;

Sequence seq4;

interleave(seq3, seq4, combSeq);

assert(combSeq.empty() == true);

cout << "Tests passed" << endl;