**CS 32 Project 2 Report**

**Design**

The doubly linked list that I implemented for project 2 was a regular linear doubly linked list with a head and tail pointer. Inside the doubly linked list (i.e. Sequence), I defined a private struct, Node, to represent all of the nodes in the doubly linked list. Each Node struct contained a “next” pointer, to point to the next node in the list, and a “prev” pointer to point to the previous pointer in the list. The doubly linked list itself contained 3 private data members: a “head” pointer (points to first Node in list), “tail” pointer (points to last node in list), and a size counter, called seq\_size. No sentinel nodes were used in this linked list, which means that an empty sequence consisted of seq\_size == 0, head == nullptr, and tail == nullptr. In a typical Sequence the prev pointer of the first node and next pointer of the last node would both be set to nullptr since there are no nodes before the first or after the second.

Here are illustrations of an empty Sequence and a typical Sequence.

**Sequence empty; Sequence typical;**

Nodes

**seq\_size == 3**

**head**

**tail**

**seq\_size == 0**

**head**

**tail**

next

prev

next

prev

Nullptrp

Nullptrp

next

prev

**Pseudocode**

Here is Pseudocode of my nontrivial functions.

\*\* empty() and size() are trivial\*\*

Sequence(){

Set head and tail to nullptr

Set seq\_size to 0

}

Sequence (const Sequence &oldseq){

Initialize sequence to an empty sequence.

If oldseq is not empty

Insert values of oldseq into the sequence at same positions

Sequence size will automatically adjust when using insert function

}

Sequence& operator= (const Sequence& otherseq){

Use copy constructor to create temporary sequence initialized to otherseq

Swap temporary sequence with sequence operated on.

Return the current sequence

}

~Sequence(){

start at the head of the sequence

while you haven’t gotten passed the last node

delete current node

advance to next node

}

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

if the position you want to insert to is within the Sequence or after the last element

if you are adding to an empty list

create a new node

set its value to value

put it in the list

increment the sequence size by 1

report success

if you are adding to the front of a nonzero Sequence

create a new node

set its value to value

place it in front of the first node

increment the sequence size

report success

if you are adding to end of a list

create a new node

set its value to value

place it after the last node

increment the sequence size

report success

if you are adding to middle of a nonzero Sequence

find desired position in Sequence

create a new node

set its value to value

place it at desired position, making sure to adjust nodes before and after

increment sequence size

report success

otherwise report failure to insert

}

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

If the Sequence is empty

Insert node with value “value” in the list using last function

Report that node was inserted in Sequence

If value is <= value of first node in list

Insert node with value “value” at the front of the list

Report that node was inserted at front

Find first node in list where value <= that node’s value

Insert node at that position

Report position where node was inserted

If value is greater that all current values in list

Insert value at end of the list

Report that a node was inserted at the end

}

bool Sequence::erase(int pos){

check if pos is a valid position in the Sequence.

If the sequence is empty

Report failure to erase anything

Elseif there is only one node

Delete that node and set sequence to empty state

Decrement size

Report success

Elseif you are deleting the front node

Reset the pointers at front

Delete that node

Decrement size

Report success

Elseif you are deleting the last node

Advance to last node

Reset the pointers at end

Delete that node

Decrement size

Report success

Elseif you are deleting a middle node

Advance to middle node

Reset the pointers in middle

Delete that node

Decrement size

Report success

Otherwise report failure

**}**

int remove (const Itemtype &value){

if sequence is empty

report no nodes removed

Starting at position 0

While you have not passed the last node

If the value at that position == value

Erase that node

Add 1 to number removed

Restart loop

Otherwise

Move to next position

Report the number of nodes deleted

}

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

if pos is a valid position

advance to node at position pos

store value at that node in “value” reference

report success

otherwise report failure to get a value

}

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

if pos is a valid position

advance to node at position pos

change value at that node to “value”

report success

otherwise report failure

}

void swap(Sequence &other){

create temporary variables to store the head,tail, of “other” and own seq\_size

set other’s head and tail to own head and tail

set own head and tail to other’s head and tail through temp variables

set other’s size to own size

set own size to other’s size

}

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

if seq2 is larger than seq1

report failure

starting at position 0 for a long as seq2 can fit in the length of seq1 advance down the length of seq1

if the first value of seq2 is == to value at current position in sequence 1

compare subsequent elements in seq2 to seq1

if they are ever not equal restart this loop

if all elements of seq2 are found sequentially in seq1

return position in seq1 where subsequence starts

otherwise report failure

}

void interleave(const Sequence& seq1, const Sequence& seq2, Sequence& result){

create a temporary sequence to represent combined sequence

starting at position 0 in seq1,seq2, and temp

while you have not gotten to end of seq1 or seq2

if you haven’t reached the end of seq1

add element at current position of seq1 into current position of temp

move up one position in seq1 and temp

if you haven’t reached the end of seq

add element at current position of seq2 into current position of temp

move up one position in seq2 and temp

once loop is finished swap temp with result (this avoids aliasing)

}

**Test Cases**

To test my code rigorously, I developed functions to run in a main sequence that would try as close as possible to test each function’s functionality independently. I realize that it is not possible to run independent tests on each member-function and non-member function that has to do with sequence, but implicitly through much code, testing, and using the debugger, the functionality of each of these functions has been verified. The dump function used at the end were used to check contents of each Sequence. All tests were run in one main program to make sure they all work together.

void test\_empty(){

Sequence empty; //construct empty sequence

assert(empty.empty()); //make sure empty sequence is empty

}

void test\_size(){

Sequence nothing;

assert(nothing.size() == 0); //make sure empty sequence has 0 size

}

void test\_insert1(){

Sequence seq1;

assert(seq1.empty() && seq1.size()==0 ); //make sure I am working with an empty sequence

assert(!seq1.insert(1, 0) && !seq1.insert(-1, 0)); //make sure I cannot insert into invalid positions

assert(seq1.insert(0, 1) && !seq1.empty() && seq1.size()==1);//make sure successful insert returns true, and seq1 is not empty and size has been incremented by 1

//test multiple insertions and insertion into end works

for(int i = 1; i<100; i++){

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

seq1.insert(i, 2);

}

assert(seq1.insert(0, 1)); //make sure insertion into beginning of non-zero sequence works

}

void test\_insert2(){

Sequence seq2;

assert(seq2.empty() && seq2.size()==0 );

assert(seq2.insert(2)==0 && seq2.size() ==1); //make sure insertion successful, in correct position, size incremented correctly

assert(seq2.insert(3)==1 && seq2.insert(0)==0); //makes sure smallest number is added before 1st element it is <=

assert(seq2.find(2) ==1); //implicitly makes sure Seqeunce contents in correct order

}

void test\_erase(){

Sequence to\_erase;

assert(!to\_erase.erase(0) && !to\_erase.erase(1) ); //makes sure can't erase a zero sequence

for( int i = 0; i<100; i++){

to\_erase.insert(0);

}

//test multiple erasures and erasing from end of Sequence

for (int j = 100; j>0; j--) {

assert(to\_erase.size() == j);

to\_erase.erase(j-1);

}

to\_erase.insert(1);to\_erase.insert(7);to\_erase.insert(12);

assert(to\_erase.erase(1)); //makes sure can erase from middle

assert(to\_erase.find(7) == -1 && to\_erase.find(12) == 1); //makes sure contents moved correctly

assert(to\_erase.erase(0)); //makes sure can erase from end

}

void test\_remove(){

Sequence to\_remove;

for( int i = 0; i<100; i++){

to\_remove.insert(0);

}

to\_remove.insert(50, 100);

assert(to\_remove.size()==101);

assert(to\_remove.remove(0)==100 && to\_remove.size()==1); //tests removal of multiple nodes

assert(to\_remove.remove(10)==0 && to\_remove.remove(100)==1); //test removal of 0 nodes and 1 node

assert(to\_remove.size()==0);

}

void test\_get(){

Sequence treasure;

ItemType loot;

//store treasure :)

assert(!treasure.get(1, loot));//tests to make sure cannot get from invalid position

treasure.insert(9);treasure.insert(0); treasure.insert(12);treasure.insert(7);

treasure.insert(1, 60);

assert(treasure.get(1, loot) && loot == 60); //tests to make sure value "got" is correct and get returns true

}

void test\_set(){

Sequence stuff;

assert(!stuff.set(5, 6)); //cannot set to invalid position

stuff.insert(0);stuff.insert(3);stuff.insert(7);stuff.insert(14);

assert(stuff.set(1, 8) && stuff.find(8) ==1 ); //tests that set works and value actually changed

}

void test\_find(){

Sequence lost\_found;

assert(lost\_found.find(0)==-1); //tests that cannot find in zero Sequence

lost\_found.insert(17);lost\_found.insert(17);lost\_found.insert(1);lost\_found.insert(0);

assert(lost\_found.find(12)==-1); //cannot find number that does not exist in Sequence

assert(lost\_found.find(17)==2); //finds 17 and reports pos where first found

}

void test\_swap(){

Sequence swap\_me;

Sequence swap\_him;

Sequence swap\_her;

swap\_me.insert(0, 6);swap\_me.insert(0, 12);

for(int i = 0; i<10; i++) {

swap\_him.insert(5);

}

swap\_me.swap(swap\_me);

assert(swap\_me.size()==2 && swap\_me.find(12)==0); //tests aliasing

swap\_me.swap(swap\_him);

assert(swap\_him.size()==2 && swap\_him.find(12)==0); //makes sure swap worked (i.e. contents and size swapped)

assert(swap\_me.size()==10 && swap\_me.find(5)==0 && swap\_me.remove(5)==10);

swap\_him.swap(swap\_her);

assert(swap\_him.size()==0 && swap\_him.find(12)==-1); //tests swapping with 0 Sequence

assert(swap\_her.size()==2 && swap\_her.find(12)==0);

}

void test\_destructor(){

Sequence destruct\_nothing;

Sequence destruct\_something;

//tested with cerr debug statement in code

for (int i = 0; i<10; i++){

destruct\_something.insert(0, 0);

}

}

void test\_copy(){

Sequence A;

Sequence B(A);

assert(A.size() == 0 && B.size() == 0); //tests copy constructor for 0 Sequence

for (int i = 0; i<100; i++) {

A.insert(1);

}

Sequence C(A);

//test copy constructor for non-zero sequence

assert(A.size() == 100 && C.size() == 100 && A.find(1) == 0 && C.find(1) == 0 );

}

void test\_assign(){

Sequence a;

Sequence b;

Sequence z;

z.insert(5);z.insert(3);z.insert(62);

a.insert(0);a.insert(1);a.insert(3);a.insert(7);a.insert(0, 12);

b=a;

a.dump(); //compare contents in a cerr dump

b.dump();

assert((a.find(12)==b.find(12))&& (a.find(7)==b.find(7))); //tests assignment to 0 Sequence

assert(a.size()==b.size());

a=a;

a.dump();

assert((a.find(12)==b.find(12))&& (a.find(7)==b.find(7))); //tests self assignment

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

z=a;

assert(z.find(62)==-1 && z.size() == 5); //tests self assignment

}

void test\_interleave(){

Sequence c;

Sequence d;

Sequence e;

c.insert(12);c.insert(2);c.insert(7);c.insert(6);

d.insert(0, 5);

d.insert(1, -3);

c.dump(); d.dump(); e.dump();

interleave(c, e, c); //test interleave with aliasing

c.dump();

assert(c.size() == 4 && c.find(2 ==0));

assert(e.size()==0);

interleave(e, e, e); //test 0 Sequence aliasing

e.dump();

assert(e.size()==0);

interleave(c, d, e); //test normal function

assert(c.size()==4 && d.size()==2 && e.size()==6);

assert(e.find(2)==0 && e.find(5)==1 && e.find(-3)==3 && e.find(12) == 5);

interleave(d, d, d); //test aliasing

assert(d.size() == 4 && d.find(-3)==2);

d.dump();

e.dump();

}

void test\_subsequence(){

Sequence p;

Sequence q;

Sequence r;

Sequence s;

assert(subsequence(p, q)==-1); //cannot find subsequence in 0 Sequences

p.insert(32);

assert(subsequence(p, q)==-1 && subsequence(q, p)==-1); //cannot find a subsequence of a trivial Sequence(q) in a nontrivial one (p)

p.insert(0, 8);p.insert(0, 29);p.insert(0, 17);p.insert(0, 63);p.insert(0, 17);p.insert(0, 42);p.insert(0, 63);p.insert(0, 21);p.insert(0, 30);

q.insert(0, 29);q.insert(0, 17);q.insert(0, 63);

p.dump();

q.dump();

assert(subsequence(p, q)==5 && subsequence(q, p)==-1); //tests normal function of subsequence and that a larger sequence cannot be found as a subsequence of a smaller sequence

r.insert(0, 32);r.insert(0, 8);r.insert(0, 29);

r.dump();

assert(subsequence(p, r)==7); //tests subsequence that goes to the last element of the main Sequence

s.insert(0, 63);s.insert(0, 21);s.insert(0, 30);

s.dump();

assert(subsequence(p, s)==0); //tests subsequence starts from first element of the main Sequence

}