**CS32 Project 2 Report by Melody Chen #705120273**

**Description of Design**

For my doubly linked list, I implemented a circular linked list with a dummy node. In each one of my node, I hold a pointer to the next node and a pointer to the previous node and the value of ItemType.

**Pseudocode**

Default Constructor

Create and initialize dummy node

Destructor

Repeatedly

Erase each node from end of list

Erase dummy node

Copy Constructor

create new dummy node

Repeatedly

Copy value from original linked list to this list

Assignment Operator

Use copy constructor to create temp copy of desired linked list

Swap this linked this with the temp

return this

Insert at designated location

check if pos is in desired range

repeatedly

go through linked list till pos

set new node with desired value to pos

shift previous and latter node

Insert

Repeatedly

Check and find correct loc for new node by comparing node values

Insert node at correct pos

Erase

check within range of list

Repeatedly

Find pos specified

Alter previous and latter node

Delete node at pos

Remove

Repeatedly

Check item in linked list = value

delete item that = value

Get

Check if pos is in range of list

Repeatedly

loop till pos in list

Copy value of node in pos to value

Set

Check is pos is in range of list

Repeatedly

loop till pos in list

Set value of node at pos to value

Find

Repeatedly

Compare value of node to value

return pos of node if equal

Swap

Swap size of list

Swap dummy nodes

Subsequence

Check if sequence 1 size is greater than sequence 2 and sequence 2 not equal 0

Repeatedly

Compare value at same position

Repeatedly

Compare rest of value

Interleave

Repeatedly

Copy one item from seq1 and one item from seq2 into result

Copy rest of item from the greater sequence into result

return result

**Test Cases**

Sequence a;

assert(a.insert(0,1)==0); *//testing insert with size 0*

assert(a.insert(0,0)==0); *//testing insert at 0*

assert(a.insert(2,3)==2); *//testing insert at 2*

assert(a.insert(2,2)==2); *//testing insert*

assert(a.insert(4,4)==4); *//testing insert*

assert(a.size()==5); *//testing size*

**unsigned** **long** test;

assert(a.get(0, test)==**true**); *//testing get*

assert(test==0); *//testing value in list*

assert(a.get(5, test)==**false**); *//testing get*

assert(a.get(4, test)==**true**); *//testing get*

assert(test==4); *//testing value in list*

assert(a.get(-1, test)==**false**); *//testing get*

assert(a.insert(6, 9)==-1); *//testing insert at invalid location*

assert(a.insert(0)==0); *//testing insert at right position*

assert(a.insert(5)==6); *//testing insert*

assert(a.insert(100)==7); *//testing insert*

assert(a.insert(1)==2); *//testing insert*

assert(a.erase(0)==**true**); *//testing erase*

assert(a.erase(0)==**true**); *//testing erase*

assert(a.erase(6)==**true**); *//testing erase*

assert(a.erase(6)==**false**); *//testing erase*

assert(a.remove(1)==2); *//testing remove*

assert(a.find(2)==0); *//testing find*

assert(a.find(6)==-1); *//test find*

assert(a.find(5)==3); *//test find*

Sequence b;

*//testing for empty list*

assert(b.remove(0)==0); *//test for empty*

assert(b.find(0)==-1); *//test for empty*

assert(b.erase(0)==**false**); *//test for empty*

assert(b.set(5, 10)==**false**); *//test for empty*

assert(b.get(0,test)==**false**); *//test for empty*

assert(b.insert(0)==0); *//test for empty*

assert(b.erase(0)==**true**); *//test for one element*

*//testing for all same values*

**for**(**int** k=0; k<4; k++)

b.insert(1);

assert(b.find(1)==0); *//test find*

assert(b.size()==4); *//test size*

assert(b.remove(1)==4); *//test remove*

assert(b.size()==0); *//test changes in size*

*//regular testing*

assert(b.insert(0)==0); *//test insert*

assert(b.insert(1)==1); *//test insert*

assert(b.insert(2)==2); *//test insert*

assert(b.set(0, 1)==**true**); *//test set*

assert(b.get(0, test)==**true**); *//test get*

assert(test==1); *//test if value changed*

assert(b.set(2,3)==**true**); *//test set*

assert(b.get(2, test)==**true**); *//test get*

assert(test==3); *//test if value changed*

assert(b.set(3,3)==**false**); *//test set*

*//testing swap*

Sequence d;

Sequence e;

ItemType v = 0;

*// No failures inserting 5 items into b*

**for** (**int** k = 0; k < 5; k++)

assert(d.insert(v) != -1);

*// When two Sequences' contents are swapped, their capacities are*

*// swapped as well:*

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

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

assert(d.find(v)==0);

assert(e.find(v)==-1);

e.swap(d);

assert(e.size()==5); *//check if their sizes are swapped*

assert(d.size()==0); *//check if sizes are swapped*

assert(d.find(v)==-1);

assert(e.find(v)==0);

assert(d.insert(v) != -1); *//check if you're able to insert stuff into b now*

assert(d.find(v)==0); *//check if the first instance of v is at 0*

*//testing copy constructor*

Sequence f(a);

assert(f.size()==a.size()); *//test if size equal*

assert(f.get(0, test)==**true**); *//test if copied*

assert(test==2); *//test if value copied*

assert(f.get(1,test)==**true**); *//test if copied*

assert(test==3); *//test if value copied*

assert(f.get(2,test)==**true**);

assert(test==4); *//test if value copied*

assert(f.get(3,test)==**true**);

assert(test==5); *//test if value copied*

assert(f.insert(0)==0); *//test if insert works*

assert(f.find(5)==4); *//test find*

*//testing assignment operator*

f = b;

assert(f.size()==3); *//test if size changed*

f.get(0, test);

assert(test==1); *//test if value changed*

f.get(1, test);

assert(test==1); *//test if value changed*

f.get(2, test);

assert(test==3); *//test if value changed*

assert(f.remove(1)==2); *//test if remove*

assert(f.size()==1); *//test if size change*

*//test subsequence*

Sequence g;

Sequence i;

Sequence z;

assert(subsequence(g, i)==-1); *//test subseq for no subseq*

g.insert(2);

g.insert(5);

g.insert(7);

g.insert(9);

g.insert(12);

g.insert(17);

i.insert(7);

i.insert(9);

i.insert(12);

z.insert(4);

z.insert(8);

z.insert(9);

z.insert(9);

assert(subsequence(g, g)==0); *//test for aliasing*

assert(subsequence(g, i)==2); *//test for subseq*

assert(subsequence(i, g)==-1); *//test for no subseq*

assert(subsequence(g, z)==-1); *//test for no subseq*

*//test interleave*

Sequence k(a);

i.insert(15);

i.insert(22);

i.insert(25);

interleave(i, z, k); *//test interleave*

k.get(0, test);

assert(test==7);

k.get(1, test);

assert(test==4);

interleave(i, z, i); *//test aliasing*

i.get(0, test);

assert(test==7); *//test if value changed*

i.get(1, test);

assert(test==4); *//test if value changed*

i.get(2, test);

assert(test==9); *//test if value changed*

i.get(3, test);

assert(test==8); *//test if value changed*

i.get(8, test);

assert(test==22); *//test if value changed*

i.get(9, test);

assert(test==25); *//test if value changed*