**Project 2 – CS 32**

**Design of doubly linked list –**

My doubly linked list is designed in a typical way. It contains a head and a tail, a next and a prev pointer variable in the nodes themselves linking each other. I do not use a dummy node nor is it a circular list.

Each node has value and a pointer to the next node and a pointer to the previous node. The node starts with head pointer and ends with the tail pointer pointing to the first and last node respectively.

**Pseudocode –**

1. Sequence :: Sequence( ) // constructor

Initialize an empty node

Head points to nullptr

Tail points to nullptr

Size = 0

1. Sequence::~Sequence() // destructor

if empty node just returns

Looping thru the entire sequence and delete each node one by one

1. Sequence::Sequence(**const** Sequence& other) // copy constructor

Copy the size of the List into a new sequence

Iterate through each node’s data and set the previous and next nodes accordingly

1. Sequence& Sequence::**operator**= (**const** Sequence& other) // assignment operator

Ensure we are not copying other onto itself

Swaps this sequence with the “other” and returns itself

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

ensures position is valid

creates a new node

checks for edge case, inserting at the beginning or end and then handles pointers respectively

if size = 0

creates a node with value and points head and tail to this node

if pos in between, loops till pos – 1 and then inserts node after this node

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

Iterates over the entire node to find the small pos where value less than value at position

Then inserts value at this pos by calling on the previous insert function

If value is greater than all values in pos then inserts value at that pos by calling the previous insert function and returns size

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

ensures position is valid

then proceeds to erase nodes at pos and sets the next and prev pointer to the correct nodes

takes care of edge cases as well, erasing the first and the last node and setting head and tail to next and previous nodes respectively

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

loop thru the list till no given value is found and keep a track of the number of iterations using count

if value is found we call erase to remove node at that position and return the pos.

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

ensures position is valid

loop over the linked list till we reach position and set p to point at node in pos

we come out of loop and set the value in that node equal to input value

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

initialize i to keep track of position

loops over the entire list till it’s not nullptr

if value at position is same as input value

return pos (stored in i)

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

create a temp variable to store current size,

switch current size with other.size and then store current size in other.size using temp.

do the same with head and tail but using a temp pointer

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

ensuring seq2 size is not 0 and greater than seq1 size

get first value of seq2

loop from 0 to seq1.size – seq2.size so that we don’t go out of bounds

get first value of seq1 and compare with seq2

if it’s a match then loop from 1 to the size of sq2 size

compare subsequent values of seq1 and seq2

if no match found

break

if match is found return the pos at which similar seq started

1. **void** concatReverse(**const** Sequence& seq1, **const** Sequence& seq2, Sequence& result)

create a temp variable

loop from seq2size to 0 so that we can store in reverse order

get values from seq2 and store in temp

loop from seq1size to 0 so that we can store in reverse order

get values from seq1and store directly in result

loop from 0 to temp size

get values from temp and insert in result

**Test Cases -**

Sequence s;

assert(s.size() == 0) // Testing an empty set

assert(s.empty() == 1) // Testing empty() function

Sequence m;

assert(m.insert(0, 10) == 0); // Test the insert(pos,value) function

assert(m.insert(1, 20) == 1);

assert(m.insert(2, 30) == 2);

ItemType value;

assert(m.get(0,value) == 1); // Test get() function

assert(m.get(3,value) == 0);

assert(m.insert("laddoo") == 0); // testing insert(value) function

assert(m.size() == 4); // test size() function

assert(m.find("laddoo") == 0); // test find() function

assert(m.remove("laddoo") == 1); // test remove() function

assert(m.erase(0) == 1); // test erase() function

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

Sequence m1;

assert(m.insert(0, "butter") == 0); // Test the swap() function

assert(m.insert(1, "paneer") == 1);

assert(m.insert(2, "naan") == 2);

m.swap(m1);

m.dump();

m1.dump();

Sequence seq1; // testing subsequence() function

seq1.insert(0, "apple");

seq1.insert(1, "banana");

seq1.insert(2, "orange");

seq1.insert(3, "pear");

seq1.insert(4, "kiwi");

Sequence seq2;

seq2.insert(0, "orange");

seq2.insert(1, "pear");

assert(subsequence(seq1, seq2) == 2);

Sequence seq3; // testing concatReverse() function

assert(seq3.insert(0, "one") == 0);

assert(seq3.insert(1, "two") == 1);

assert(seq3.insert(2, "three") == 2);

Sequence seq4;

assert(seq4.insert(0, "hello") == 0);

assert(seq4.insert(1, "world") == 1);

// seq3.dump();

//

// seq4.dump();

Sequence result;

concatReverse(seq4, seq3, result);

// result.dump();

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

assert(result.find("world") == 0);

assert(result.find("hello") == 1);

assert(result.find("three") == 2);

assert(result.find("two") == 3);

assert(result.find("one") == 4);