**Linked List Design**

For my project, I implemented a circular doubly linked list. Each node consists of 3 variables: one of type ItemType that keeps track of the current value in the node, another is a pointer that points to the next node in the list, and the last is a pointer that points to the node directly before the current node. My linked list also features a dummy node, so instead of having the head point to the first meaningful node in the list, it points to a node that then points to the first and last node in the list. In addition, since my list is circular, the last node’s next pointer points to the dummy node, not nullptr. Lastly, each node is inserted into the list in ascending order based on it’s value, and because of this, my list always maintains sorted order where the first element hold the smallest value and the thast node holds the largest value.

**Pseudocode**

Insert()

If length is 0

Create a new node

Insert it directly after dummy node

Else

Loop through linked list

If current Node's value > value being inserted

Create a new node

Reassign surrounding node’s pointer values

Insert new node before the current node

Copy Constructor()

If length is 0

Insert only a dummy node

Else

Loop through other Set’s list

Insert each s2 node value into current linked list

Assignment Operator Overload()

Check for aliasing

Create a copy of rhs Set

Swap current value with the copy

Destructor()

Loop through list

Delete current Node

Point to the next node

Contains()

Loop through lists

If current value = desired value

Return true

Erase()

If the value is in the list

Loop through the list

If the current node equals the desired node

Reassign the pointer’s of the node directly before and after the node

Remove current node

Return true

Get()

if desired position exits within the list

Loop through linked list till num iterations = desired node

Set current element equal to next node

Set return value equal to the current node’s value

Swap()

Create temp variables that stores items within Set

Set current items equal to other Set’s items

Set other set’s items equal to temp items

Unite()

Create a copy of s1

Loop through s2

Get value of s2 at current index

Insert current value into s1 copy

Set result equal to the s1 copy

Subtract()

Create a copy of s1

Loop through s2

Get current value of s2

If s1 contains current value

Erase current value from s1 copy

Else

Insert current value into s1 copy

Set result equal to the s1 copy

**Test Cases**

Set s1; //Call the default constructor

assert(s1.size() == 0); //make sure the default constructor doesn't add any meaningful items

assert(s1.insert("item1")); //Checks to make sure that elements can be inserted

assert(s1.insert("item2")); //

assert(s1.insert("item6")); //

assert(s1.insert("item4")); //

assert(s1.insert("item5")); //

assert(s1.insert("item3")); //

assert(s1.size() == 6); //Checks to make sure that the insert() function properly keeps track of the size

assert(!s1.insert("item1")); //Makes sure that only unique elements can be added

assert(!s1.insert("item3")); //

ItemType temp;

assert(s1.get(0, temp) && temp == "item1" && s1.get(3, temp) && temp == "item4"); //checks to make sure get works properly, and that the values are stored in proper order

assert(s1.erase("item5") && s1.erase("item6") && s1.erase("item2")); //Checks to make sure delete returns the right value

assert(s1.size() == 3); //checks to make sure that erase actually decreases the length

assert(!s1.contains("item5") && !s1.contains("item6")); //makes sure that the items were actually deleted

assert(s1.insert("item2"));

//s1.dump(); //1,2,3,4

///////////////////////////TESTS INSERT,GET,SIZE,ERASE,CONTAINS

Set s2;

assert(s2.empty());

s2 = s1; //Calls the copy constructor which has not been tested yet

assert(s2.size() == s1.size()); //make sure that the two elements are the same size

for(int i = 0; i < s2.size(); i++)

{

ItemType t1;

ItemType t2;

s1.get(i, t1); // these two values should be the same

s2.get(i, t2);//

assert(t1 == t2);//Makes sure that each Set's linked list is the same, and that the copy constructor worked like intended

}

assert(s1.insert("randomItem")); //insert a value into s1

assert(!s2.contains("randomItem")); //make sure that s2 doesn't contain the new item, shows that pointers were initialized properly

assert(s2.insert("newItem")); //same as above

assert(!s1.contains("newItem"));

Set s3, s4;

assert(s3.insert("a"));

assert(s3.insert("b"));

assert(s3.insert("c"));

assert(s4.insert("1"));

assert(s4.insert("2"));

assert(s4.insert("3"));

s3.swap(s4); //check to make sure that swap function works

ItemType newT;

assert(s3.get(0, newT) && newT == "1" && s3.get(1, newT) && newT == "2" && s3.get(2, newT) && newT == "3" );

assert(s4.get(0, newT) && newT == "a" && s4.get(1, newT) && newT == "b" && s4.get(2, newT) && newT == "c" );

assert(s3.insert("0") && !s4.contains("0")); //makes sure that pointers are properly swapped

//The above assert statements make sure that each element of each list was properly swapped, and that each list only consists of the values from the list it swapped with

Set s5;

s5.insert("random");

s5 = s1; //This calls the assignment operator

assert(s5.size() == s1.size()); //shows that each set is the same length as expected

for(int i = 0; i < s5.size(); i++)

{

ItemType t1;

ItemType t2;

s1.get(i, t1);

s5.get(i, t2);

assert(t1 == t2); //check to make sure that each set now consists of the same items

}

assert(s5.insert("item0") && !s1.contains("item0")); //makes sure that each Set is separate from each other

//Shows that inserting an element only inserts into the proper list, and not the one that was used to copy elements

////////////////////TESTS ASSIGNMENT

Set s6;

s6.insert("1");

s6.insert("2");

s6.insert("3");

Set s7;

s7.insert("2");

s7.insert("11");

s7.insert("12");

s7.insert("13");

unite(s6, s7, s6); //Calls the unite function

assert(s6.contains("1") && s6.contains("11")); //shows that even if the third list is the same as one of the input lists, the third list consists of the right elements, and is not deleted

//s6.dump();

Set s8;

assert(s8.insert("random"));

unite(s6, s7, s8); //tests unite with 3 unique Set's

assert(s8.contains("1") && s8.contains("11") && !s8.contains("random")); //shows that even though the third list had items in it, they no longer exist. The result list only consists of elements from the first two Set's

//////////////////TESTS UNITE

Set s9,s10,s11;

s9.insert("a");

s9.insert("b");

s9.insert("c");

s9.insert("d");

s11.insert("a");

s11.insert("b");

s11.insert("c");

s11.insert("e");

//subtract(s9, s11, s9);

subtract(s9, s11, s10); //Calls subtract on 3 unique Set's

assert(!s10.contains("a") && s10.size() == 2 && s10.contains("e") && s10.contains("d"));

//Shows that the result list only consists of unique elements from each list as expected

assert(s11.insert("remaining"));

assert(s11.insert("d") && s11.erase("e"));

subtract(s9, s11, s10); //Call subtract with 3 lists, but this time the result list is not empty

assert(s10.size() == 1 && s10.contains("remaining")); //shows that the result list ONLY consists of unique elements, and any of the data previously stored it it is no longer there

cerr << "FINISHED" << endl; //Shows that the program properly executes, if this is printed, it means that all of the asserts are passed