Doubly Linked List Implementation:

The list is a doubly linked list containing no head nor tail pointers, instead containing a head dummy node which is a private member of the Set class. The list is accessed using this node.

Each list node consists of:

1. m\_value, a value of ItemType, ItemType being any variable type
2. m\_next, a pointer to the next Node
3. m\_prev, a pointer to the previous node

The last node in the list is identified by its m\_next pointer being equal to nullptr.

Non-trivial algorithms:

Set operator=(const Set& src):

* initializes set as copy of ‘src’
* swaps current set with copy
* copy is automatically destructed when functions call ends

bool get(int pos, ItemType& value):

* upon insertion, values input into a set are sorted with the first node being < all other nodes
* if ‘i’ is within bounds, iterates to the ith item which satisfies conditions of ‘get’.

void unite(const Set& s1, const Set& s2, Set& result): (set result becomes a set containing all values within s1 and s2 and no others)

* result is set equal to s1, simultaneously erasing any values within result already
* repeating as many times as the size of s2:
  + uses get to get a new value from s2
  + attempts to insert into result (if value is already there, due to restrictions of a set, it will not be added)

four special cases:

s1 and s2 are same

result = s1

no issues from adding s2 values (nothing will actually be added)

s1 and result are same

result = s1 does nothing

values from s2 are added to result

s2 and result are same

special case in code, where instead of adding values from s2 adds values from s1

s1 s2 and result are same

result = s1 does nothing

adding values from s2 does nothing

void butNot(const Set& s1, const Set& s2, Set& result): (set results becomes set containing all values within s1 and s2)

s1 and s2 are same/ s1 s2 and result are same

result is set to empty

s2 and result are same

makes a temporary copy of s2

sets result = s1

removes any values in s2 copy from result

s1 and result are same or all unique values

result = s1 (does nothing if result and s1 are same)

s2 already equals result

makes a temporary storage for what is within s2 already

result = s1

remove temporary storage

result is set to empty

tests:

// // test functions!

// using Item Type = std::string

Set s1;

// checks that freshly initialized set is empty

assert(s1.size() == 0 && s1.empty());

ItemType a = "unchanged";

// check that cannot remove random values and nothing to get

assert(s1.erase("h") == false && s1.get(0, a) == false && a == "unchanged");

// checking insert and size functions

assert(s1.insert("a") && s1.size() == 1 && s1.empty() == false);

// cannot insert previously inserted values

assert(s1.insert("a") == false);

// adding new values

assert(s1.insert("b") && s1.insert("c") && s1.insert("d"));

assert(s1.size() == 4);

// testing get works properly

assert(s1.get(2, a) && s1.contains(a) && a == "c");

Set s2 = s1;

Set s3;

s3 = s1;

// copy constructor and assignment operators work properly

assert(s2.contains("d") && s2.size() == 4);

assert(s3.contains("c") && s3.size() == 4);

Set s4;

s4.insert("z");

s4.insert("y");

// testing swap

s4.swap(s1);

assert(s1.contains("z") && s1.contains("a") == false && s1.size() == 2);

assert(s4.contains("b") && s4.contains("y") == false && s4.size() == 4);

// test erase function, from both front, middle, and back of sorted set

assert(s4.erase("a") && s4.erase("c"));

assert(s4.get(2, a) == false && s4.get(0, a));

assert(a == "b");

assert(s4.erase("d") && s4.erase("b"));

// s1 now empty

Set empty1;

s1.swap(empty1);

s1.insert("a");

s1.insert("b");

s1.insert("c");

s2 = s1;

s2.insert("d");

s2.insert("e");

s2.insert("f");

// test of butnot

butNot(s2, s1, s3);

assert(s3.contains("a") == false && s3.contains("d") && s3.size() == 3);

// s3 now contains d, e, f

// test of unite

unite(s1, s3, s4);

assert(s4.contains("a") && s4.contains("b") && s4.contains("c") && s4.contains("d") && s4.contains("e") && s4.contains("f") && s4.contains("z") == false && s4.size() == 6);