Professor Smallberg – Lec 2 Leo Gretzinger

Project 2 Report 1/28/19

**Doubly-linked list:**

I made my doubly-linked list circular and with a dummy node. Each node in the list is a object of the Node struct with data members: data, next (to point to the next node) and prev (to point to previous node). I tried to sort the values in the set during insertion but failed in one small area – as a result the Nodes in the set are order first added to most recently added (left to right).

**Non-trivial Pseudocode:**

*Assignment operator:*

*If the two sides of the operator are not at the same*

*Delete the set meant to change*

*Create a new empty set*

*Loop through and assign the values from the old set to the new set*

*Return a pointer to the new set*

*Insert function:*

*If the value to be inserted doesn’t already exist in the set*

*Create a new node and assign the intended value to it*

*Adjust the set (the new node points forward to the dummy and back to the previous tail, the previous tail points forward to the new node and the dummy node points back to the new node)*

*Increment size and tail pointer*

*Erase function:*

*Loop through the set to see if the intended value exists within it*

*If it doesn’t return false, if it does, continue*

*Adjust the set (the pointer behind the target node now points forward to the pointer ahead of the target node and vice versa)*

*Decrement size, adjust tail if the end node was removed and delete the node erased*

*Get function:*

*Loops through the set with inner and outer loop*

*Outer loop has specified node and inner loop counts how many nodes are less than the specified node*

*If the count is equal to the specified count, break out and set the value equal to the specified node’s data*

*Unite function:*

*Assigns one set to the result set*

*Loops through the other set and inserts (with above function) any new values into the result set*

*The result set now contains all the values within both sets, without repetitions*

*Subtract function:*

*Assigns first set to the result set*

*Loops through second set and erases any common items between the two sets (within result)*

*The result set now contains only the unique values in the first set*

**Tests:**

int main() {

Set s; // default constructor

assert(s.empty()); // empty function

assert(s.size() == 0); // tests size function when size = 0

assert(!s.erase("o")); // erase empty is false

assert(s.insert("y"));

assert(s.insert("d"));

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

assert(s.insert("e"));

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

assert(s.size() == 5); // tests size function when size > 1

assert(s.insert("g"));

assert(s.insert("q"));

assert(!s.insert("q")); // can't repeat insert

assert(s.insert("p"));

s.dump(); // Insert 8x and dump to check correct values

assert(s.contains("y")); // contains in beginning

assert(s.contains("p")); // contains at end

assert(s.contains("e")); // contains in middle

assert(!s.contains("h")); // does not contain

string str = "bleh";

assert(!s.get(8, str) && str == "bleh"); // No change to str if i is to big in get

assert(s.get(0, str) && str == "b"); // str is smallest val if i = 0 after get

assert(s.get(3, str) && str == "e"); // str is middle val if i = 3 after get

assert(s.erase("y")); // Erase first value

assert(s.erase("p")); // Erase end value

assert(s.erase("e")); // Erase middle value

s.dump(); // Erases and dumps to check correct values

Set s2;

assert(s2.insert("e"));

assert(s2.insert("b"));

assert(s2.insert("g"));

s2.dump();

Set s4(s2);

s4.dump(); // Copy constructor and dump to check

s4 = s;

s4.dump(); // Assignment operator and dump to check

Set s3;

Set s5;

unite(s2, s, s3);

s3.dump(); // Unite with smaller then larger (and dump applies to all)

unite(s, s2, s3);

s3.dump(); // Unite with larger then smaller (also tests if works when s3 not-empty to start)

unite(s2, s5, s3);

s3.dump(); // Unite with empty and non-empty

subtract(s2, s, s3);

s3.dump(); // subtract larger from smaller

subtract(s, s2, s3);

s3.dump(); // subtract smaller from larger

subtract(s2, s5, s3);

s3.dump(); // subtract empty from non-empty

subtract(s5, s2, s3);

s3.dump(); // subtract non-empty from empty

s4.swap(s2);

s4.dump();

s2.dump(); // Swap bigger with smaller and dump

s4.swap(s2);

s4.dump();

s2.dump(); // Swap smaller with bigger and dump

s5.swap(s);

s5.dump();

s.dump(); // Swap empty with not empty and dump

cout << "All tests Passed" << endl;

} // Breakpoint here and debug to check destructor