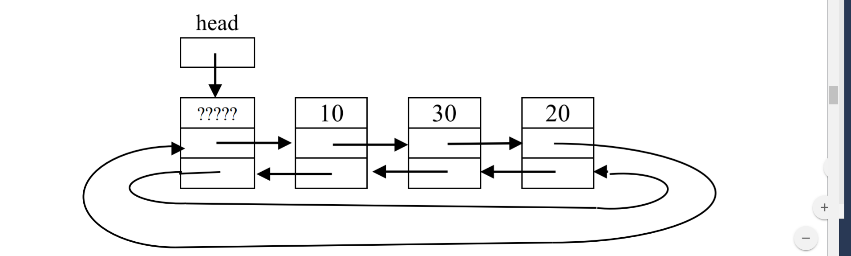
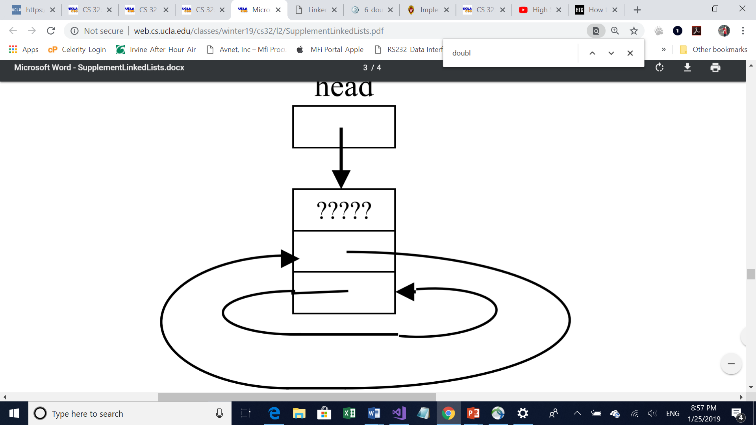
1. My implementation utilizes a circular doubly-linked list. It also has a dummy node, similar to the diagram in page 3 of the supplemented lecture notes on linked lists, as well as a head pointer that pointed to the dummy node.





Each node contains an ItemType value to store a string or unsigned long, a next pointer that points to the next node and a previous pointer that points to the previous node. They are in no particular order; the one most recently added is at the rear, while the first item is always right after the dummy node, which will always be first and created in an empty set.

1. Pseudocode for Unite:

If size of s1 is 0

set result to s2 and return

or set result to s1 and return if s2 is size 0

repeatedly:

call get for s2 to get ith smallest value

if value is not found in result,

insert value in result

Pseudocode for Subtract:

If size of s1 or s2 is 0,

set result to s2 and return

repeatedly:

call get for s2 to get ith smallest value

if value is found in result,

erase value from result

1. Test:

The tests were performed on a set of strings (i.e., ItemType was a type alias for std::string).

// default constructor

Set ss;

// For an empty set:

assert(ss.size() == 0); // test size

assert(ss.empty()); // test empty

assert(!ss.erase("roti")); // nothing to remove

//for string

Set ss;

ss.insert("lavash");

ss.insert("roti");

ss.insert("chapati");

ss.insert("injera");

ss.insert("roti");

ss.insert("matzo");

ss.insert("injera");

assert(ss.size() == 5); // duplicate "roti" and "injera" were not added

string x;

ss.get(0, x);

assert(x == "chapati"); // "chapati" is greater than exactly 0 items in ss

ss.get(4, x);

assert(x == "roti"); // "roti" is greater than exactly 4 items in ss

ss.get(2, x);

assert(x == "lavash"); // "lavash" is greater than exactly 2 items in ss

//to test swap function

Set ss1;

ss1.insert("laobing");

Set ss2;

ss2.insert("matzo");

ss2.insert("pita");

ss1.swap(ss2);

assert(ss1.size() == 2 && ss1.contains("matzo") && ss1.contains("pita") &&

ss2.size() == 1 && ss2.contains("laobing"));

//empty string case

Set ss;

ss.insert("dosa");

assert(!ss.contains(""));

ss.insert("tortilla");

ss.insert("");

ss.insert("focaccia");

assert(ss.contains(""));

ss.erase("dosa");

assert(ss.size() == 3 && ss.contains("focaccia") && ss.contains("tortilla") &&

ss.contains(""));

string v;

assert(ss.get(1, v) && v == "focaccia");

assert(ss.get(0, v) && v == "");

//more advanced testing

Set ss; //default constructor

assert(ss.insert("roti")); //testing insert

assert(ss.insert("pita")); //testing insert

assert(!ss.insert("pita")); //should not insert

assert(ss.size() == 2); //testing size

assert(ss.contains("pita")); //testing contains

ItemType x = "laobing";

assert(ss.get(0, x) && x == "pita"); //testing get in beginning case

assert(ss.get(1, x) && x == "roti"); //testing get in end case

assert(ss.erase("roti") && ss.size() == 1 && ss.get(0, x) && x == "pita"); //testing erase as well as

//new result of erase.

assert(ss.size() == 1);

Set tt;

assert(tt.empty()); //testing empty

assert(tt.size() == 0); //test size

assert(!tt.erase("laobing")); //nothing to remove

assert(tt.insert("laobing")); //test insert

assert(tt.insert("roti")); //test insert for bigger list

assert(tt.insert("bread")); //test insert again

assert(tt.size() == 3); //test size for bigger list

ss.swap(tt); //testing swap

assert(ss.size() == 3 && tt.size() == 1);

assert(ss.contains("laobing") && tt.contains("pita"));

assert(ss.get(1, x) && x == "laobing"); //test get for middle case

cerr << "done with swap" << endl;

Set result;

unite(ss, tt, result); //testing unite, which will indirectly test copy constructor

cerr << "result: " << result.size() << endl;

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

assert(result.contains("pita") && result.contains("laobing") && result.contains("roti") && result.contains("bread"));

result = ss; //testing assignment operator and get

assert(result.size() == 3);

assert(result.get(0, x) && x == "bread" && result.get(1, x) && x == "laobing"

&& result.get(2, x) && x == "roti");

Set temp = result; //copy constructor test

cerr << "created tempt from result" << endl;

assert(temp.insert("food") && temp.insert("carb") && temp.size() == 5);

unite(ss, tt, temp); //unite for non empty result

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

cerr << "testing subtract" << endl;

Set uu; //testing subtract along with nonempty result

assert(uu.insert("bread"));

subtract(ss, uu, result); //testing when result is not empty

cerr << "result.size(): " << result.size() << endl;

assert(result.size() == 2 && !result.contains("bread"));