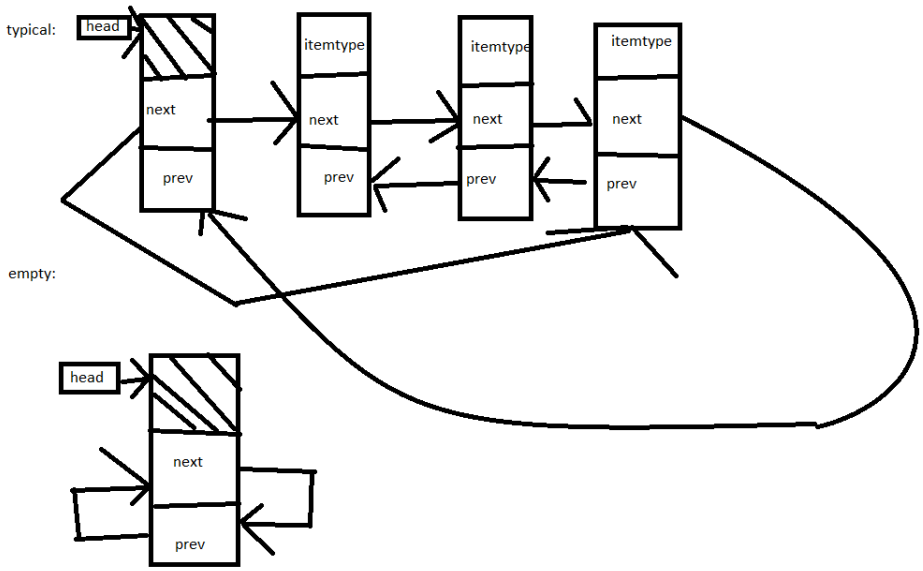
Attiano Purpura-Pontoniere

504-847-318

CS32 Project 2 Report

Description of doubly Linked-List implementation:

I implemented my Set class with a circular doubly linked list head pointing to a dummy node. Here is my Pablo Picasso of a typical set and empty set:



Psuedocode for algorithims:

Non-trivial algorithms: (subtract, unite,swap,get,erase,insert,contains,copy constructor, assignment operator)

Subtract:

Create a copy of set1 & set2

Record the size of set 2

Set result equal to the copy of set1

Repeatedly for the entirety (size) of s2:

Check if an item in s2 is already in result(set 1)

If a match is found

Delete the match from result

end

Unite:

Determine if the input result set is the same as set1 or set2

If result and set1 are the same size

If result and set 1 have all the same elements

Record that set1 is the same as result

Repeat the same steps for set2

If result Set is the same Set as Set1orSet2

If result is the same as set1

Repeatedly for the entirety (size) of set2:

Check if an item in s2 is already in result(set 1)

If not then add the item to result

If result is the same as set2

Repeatedly for the entirety (size) of set1:

Check if an item in s1 is already in result(set 2)

If not then add the item to result

If result is not the same set as either set 1 or set2

If result isn't empty

erase all the items in result

repeatedly for the size of set1:

if an item in set1 doesnt equal any of the items in set2

add the item to now-empty result set

repeatedly for the size of set2:

if an item from set2 doesnt equal any of the items in result

add the item to now potentially semi-filled result set

end

Swap:

Create a temp to hold the head pointer for this set

Swap the head pointer for this set and the head pointer for the other set

Create a temp to hold the size of this set

Swap the size of this set with the size of the other set

end

Get:

if the input index, i, is non-negative and less than the size of this set

create a counter variable for how many items a given item is greater than

repeatedly for the entire list:

reset the counter variable

check an item in the list with every other item in the list

if the item being checked is greater than another item

increment counter variable

if the count equals the input index, i

set the input value equal to the value of the item being checked

return true

otherwise return false

Erase:

If the list is empty

Return false

If the list contains the value input

Create a pointer to the head of the list

Repeatedly for the size of the list:

Move the pointer to the next item of the list

If the pointer points at head

Return false

Make the node previous the node pointed to in the list point to the node ahead of the ----node pointed to in the list

Make the node ahead of the node pointed to in the list point to the node previous the ----node pointed to in the list

Delete the node that points to the value

Decrement the size of the list

Return true

Otherwise return false

Insert:

if the list doesn't already contain the input value

create a pointer to the end of the list

create a new node with the value input

make the new node's previous pointer point to the end of the list

make the end of the list's next pointer point to the new node

make the head pointer's previous pointer point to the new node

make the new node's next pointer point to the head of the list

increment the size of the list

return true

otherwise return false

Contains:

Repeatedly for the size of the list:

If a match is found between an item in the list and input value

Return true

Otherwise return false

Copy constructor:

Create a dummy head node for this set

Store the head pointer in a temp

Repeatedly for the size of the other set:

Store a pointer to the next item to be looked at in other set in a temp-p

Store the data from each item in a temp

Create a new node with the value from temp-data

Append the new node to this set by re-arranging stored pointers to nodes

(make this set's next pointer point to the node being looked at

make this set's head's previous pointer point to the node being looked at

make the node being looked at's next pointer point to head

make the node being looked at's previous pointer point to temp-p)

move the temp-pointer forward to the next item in the list to be checked

Otherwise if the set being constructed from is empty

Make head's next and previous pointer point to itself

Assignment operator:

If the set on the left-hand-side of the equal sign isn't the same as that on the right-hand side

Create a copy of the right-hand-side set and store it in a temp set

Swap this set (left-hand-side) with the temp set

Otherwise return the set on the left-hand-side (this set)

List of test cases:

I comment why I did certain test cases, sometimes in a header of all the following tests, each time I redefine variables is a new set of tests, I would just comment out the old redefinitions and compile just a single test at a time:

```
void f(Set g) { //making sure the copy constructor acts as expected and doesnt leak
memory, which I checked for using an intwrapper class
	for (int i = 0; i < g.size(); i++) {
		ItemType x;
		g.get(i, x); //also checking that get works properly
		cerr << x << endl;
	}
}

void test()
{

	Set ss;
	assert(!ss.contains("")); //making sure contains works as expected
	ss.insert("tortilla");
	ss.insert("");
	ss.insert("dosa");
	ss.insert("focaccia");
	assert(ss.contains(""));
	ss.erase("dosa");
	assert(ss.size() == 3 && ss.contains("focaccia") && ss.contains("tortilla") &&
//making sure size returns correctly
		ss.contains(""));
	string v;
	assert(ss.get(1, v) && v == "focaccia"); //making sure get works properly as well
	assert(ss.get(0, v) && v == "");
	Set s;
	s = ss; //making sure assignment operator works properly
	assert(s.size() == 3 && s.contains("focaccia") && s.contains("tortilla") &&
		s.contains(""));*/
	Set s;
	assert(!s.erase("jim")); //making sure erase works correctly when trying to erase
from an empty list
	assert(s.empty()); //testing empty
	s.insert("jim");
	assert(!s.insert("jim")); //no duplicates possible
	assert(s.size() == 1 && s.contains("jim")); //testing size and contains
	s.erase("jim"); //testing delete for a 1 item list
	assert(s.size() == 0 && s.empty());
	s.insert("jim");
	s.insert("false");
	s.insert("king");
	Set ss;
	assert(ss.empty());
	ss = s; //making sure assignment operator works as expected
	assert(ss.size() == s.size() && s.size() == 3 && ss.contains("jim") &&
ss.contains("king") && ss.contains("false"));
	ss.erase("false");
	assert(ss.size() != s.size() && ss.size() == 2 && ss.contains("jim") &&
ss.contains("king"));
	s.swap(ss); //making sure swap works for non empty sets
	assert(s.size() != ss.size() && s.size() == 2 && s.contains("jim") &&
s.contains("king"));
	assert(ss.size() == 3 && ss.contains("jim") && ss.contains("king") &&
ss.contains("false"));
	Set j;
```

```
	assert(j.empty());
	ss.swap(j); //making sure swap works for swapping with an empty set
	assert(ss.empty() && j.size() == 3 && j.contains("jim") && j.contains("king") &&
j.contains("false"));

	f(s);
	f(ss);//making sure copy constructor works by looking at output

	// TESTING TO MAKE SURE UNITE WORKS FOR ANALISING INPUT
Set x;
	Set y;
	Set result;

	x.insert("a");
	x.insert("b");
	x.insert("c");
	x.insert("d");

	y.insert("a");
	y.insert("b");
	y.insert("c");
	y.insert("d");


	unite(x, y, x);
	assert(x.size() == 4 && x.contains("a") && x.contains("b") && x.contains("c") &&
x.contains("d"));

	unite(x, x, x);
	assert(x.size() == 4 && x.contains("a") && x.contains("b") && x.contains("c") &&
x.contains("d"));

	unite(y, y, x);
	assert(x.size() == 4 && x.contains("a") && x.contains("b") && x.contains("c") &&
x.contains("d"));
Set x;
	Set y;
	Set result;

	x.insert("a");
	x.insert("b");
	x.insert("c");
	x.insert("d");

	y.insert("a");
	y.insert("b");
	y.insert("c");
	y.insert("d");


	unite(x, y, x);
	assert(y.size() == 4 && x.contains("a") && y.contains("b") && y.contains("c") &&
y.contains("d"));

	unite(x, x, x);
	assert(y.size() == 4 && x.contains("a") && y.contains("b") && y.contains("c") &&
y.contains("d"));
```

```
	unite(y, y, x);
	assert(y.size() == 4 && x.contains("a") && y.contains("b") && y.contains("c") &&
y.contains("d"));


	Set x;
	Set y;
	Set result;

	x.insert("a");
	x.insert("b");
	x.insert("c");
	x.insert("d");

	y.insert("a");
	y.insert("b");
	y.insert("c");
	y.insert("d");

	result.insert("a");
	result.insert("b");
	result.insert("c");
	result.insert("d");

	unite(x, y, result);
	assert(result.size() == 4 && result.contains("a") && result.contains("b") &&
result.contains("c") && result.contains("d"));

	unite(x, s, result);
	assert(result.size() == 7 && result.contains("a") && result.contains("b") &&
result.contains("c") && result.contains("d") && result.contains("focaccia") &&
result.contains("tortilla") &&
		result.contains(""));

	unite(ss, y, result);
	assert(result.size() == 7 && result.contains("a") && result.contains("b") &&
result.contains("c") && result.contains("d") && result.contains("focaccia") &&
result.contains("tortilla") &&
		result.contains(""));


	//TESTING TO MAKE SURE UNITE WORKS FOR NON-ALIASING CASES WITH AND WITHOUT
DUPLICATES, WHEN SIZE 1> SIZE2 && WHEN SIZE 2> SIZE1
	//&& ALL THOSE CASES WHEN RESULT IS BOTH EMPTY AND WITH DIFFERENT NUMBERS OF
ELEMENTS IN IT

	Set x;
	Set y;
	Set result;

	x.insert("a");
	x.insert("g");
	x.insert("e");
	x.insert("d");

	y.insert("a");
	y.insert("c");
	y.insert("d");
```

```
        result.insert("k");
        result.insert("l");
        result.insert("c");
        result.insert("d");

        unite(x, y, result);
        assert(result.size() == 5 && result.contains("a") && result.contains("g") &&
result.contains("c") && result.contains("d")&&result.contains("e"));

        Set x;
        Set y;
        Set result;

        x.insert("g");
        x.insert("e");

        y.insert("a");
        y.insert("c");
        y.insert("d");

        result.insert("k");
        result.insert("l");
        result.insert("c");
        result.insert("d");

        unite(x, y, result);
        assert(result.size() == 5 && result.contains("a") && result.contains("g") &&
result.contains("c") && result.contains("d")&&result.contains("e"));*/

        Set x;
        Set y;
        Set result;

        x.insert("a");
        x.insert("g");
        x.insert("e");
        x.insert("d");

        y.insert("a");
        y.insert("c");
        y.insert("d");
        y.insert("k");


        result.insert("k");
        result.insert("l");
        result.insert("c");
        result.insert("d");

        unite(x, y, result);
        assert(result.size() == 6 &&result.contains("k") && result.contains("a") &&
result.contains("g") && result.contains("c") && result.contains("d") &&
result.contains("e"));*/

        Set x;
        Set y;
        Set result;
```

```
	x.insert("a");
	x.insert("g");
	x.insert("e");
	x.insert("d");

	y.insert("a");
	y.insert("c");
	y.insert("d");

	unite(x, y, result);
	assert(result.size() == 5 && result.contains("a") && result.contains("g") &&
result.contains("c") && result.contains("d") && result.contains("e"));*/
Set x;
Set y;
Set result;

x.insert("a");
x.insert("g");
x.insert("e");
x.insert("d");

y.insert("a");
y.insert("c");
y.insert("d");

result.insert("k");
result.insert("l");


unite(x, y, result);
assert(result.size() == 5 && result.contains("a") && result.contains("g") &&
result.contains("c") && result.contains("d") && result.contains("e"));*/

Set x;
Set y;
Set result;

x.insert("g");
x.insert("e");

y.insert("a");
y.insert("c");
y.insert("d");

result.insert("k");

unite(x, y, result);
assert(result.size() == 5 && result.contains("a") && result.contains("g") &&
result.contains("c") && result.contains("d") && result.contains("e")); */

}



void test2() {//testing all the normal functions and unite for unsigned long

	Set s;
```

```
	assert(!s.erase(20)); //making sure erase works correctly when trying to erase
from an empty list
	assert(s.empty()); //testing empty
	s.insert(100);
	assert(!s.insert(100)); //no duplicates possible
	assert(s.size() == 1 && s.contains(100)); //testing size and contains
	s.erase(100); //testing delete for a 1 item list
	assert(s.size() == 0 && s.empty());
	s.insert(100);
	s.insert(0);
	s.insert(200);
	Set ss;
	assert(ss.empty());
	ss = s; //making sure assignment operator works as expected
	assert(ss.size() == s.size() && s.size() == 3 && ss.contains(100) &&
ss.contains(200) && ss.contains(0));
	ss.erase(0);
	assert(ss.size() != s.size() && ss.size() == 2 && ss.contains(100) &&
ss.contains(200));
	s.swap(ss); //making sure swap works for non empty sets
	assert(s.size() != ss.size() && s.size() == 2 && s.contains(100) &&
s.contains(200));
	assert(ss.size() == 3 && ss.contains(100) && ss.contains(200) && ss.contains(0));
	Set j;
	assert(j.empty());
	ss.swap(j); //making sure swap works for swapping with an empty set
	assert(ss.empty() && j.size() == 3 && j.contains(100) && j.contains(200) &&
j.contains(0));

	f(s);
	f(ss);//making sure copy constructor works by looking at output

	// TESTING TO MAKE SURE UNITE WORKS FOR ANALISING INPUT

	Set x;
	Set y;
	Set result;

	x.insert(1);
	x.insert(2);
	x.insert(3);
	x.insert(4);

	y.insert(1);
	y.insert(2);
	y.insert(3);
	y.insert(4);

	result.insert(5);
	result.insert(6);
	result.insert(7);
	result.insert(8);

	unite(x, y, result);
	assert(result.size() == 4 && result.contains(1) && result.contains(2) &&
result.contains(3) && result.contains(4));

	unite(x, s, result);
```

```
	assert(result.size() == 7 && result.contains(1) && result.contains(2) &&
result.contains(3) && result.contains(4) && result.contains(0) && result.contains(100) &&
	result.contains(200));

	unite(ss, y, result);
	assert(result.size() == 6 && result.contains(1) && result.contains(2) &&
result.contains(3) && result.contains(4) && result.contains(100) &&
		result.contains(200));*/

	//TESTING TO MAKE SURE UNITE WORKS FOR NON-ALIASING CASES WITH AND WITHOUT
DUPLICATES, WHEN SIZE 1> SIZE2 && WHEN SIZE 2> SIZE1
	//&& ALL THOSE CASES WHEN RESULT IS BOTH EMPTY AND WITH DIFFERENT NUMBERS OF
ELEMENTS IN IT

	Set x;
	Set y;
	Set result;

	x.insert(1);
	x.insert(5);
	x.insert(3);
	x.insert(4);

	y.insert(1);
	y.insert(2);
	y.insert(3);
	y.insert(4);

	result.insert(5);
	result.insert(6);
	result.insert(7);
	result.insert(8);

	unite(x, y, result);
	assert(result.size() == 5 && result.contains(1) && result.contains(2) &&
result.contains(3) && result.contains(4) && result.contains(5));

	Set x;
	Set y;
	Set result;

	x.insert(1);
	x.insert(5);
	x.insert(4);

	y.insert(1);
	y.insert(2);
	y.insert(3);
	y.insert(4);

	result.insert(5);
	result.insert(6);
	result.insert(7);
	result.insert(8);

	unite(x, y, result);
	assert(result.size() == 5 && result.contains(1) && result.contains(2) &&
result.contains(3) && result.contains(4) && result.contains(5));
```

```
        Set x;
        Set y;
        Set result;

        x.insert(1);
        x.insert(5);
        x.insert(3);
        x.insert(4);

        y.insert(1);
        y.insert(2);
        y.insert(3);

        result.insert(5);
        result.insert(6);
        result.insert(7);
        result.insert(8);

        unite(x, y, result);
        assert(result.size() == 5 && result.contains(1) && result.contains(2) &&
result.contains(3) && result.contains(4) && result.contains(5));

        Set x;
        Set y;
        Set result;

        x.insert(1);
        x.insert(5);
        x.insert(3);
        x.insert(4);

        y.insert(1);
        y.insert(2);
        y.insert(3);
        y.insert(4);


        unite(x, y, result);
        assert(result.size() == 5 && result.contains(1) && result.contains(2) &&
result.contains(3) && result.contains(4) && result.contains(5));
        Set x;
        Set y;
        Set result;

        x.insert(1);
        x.insert(5);
        x.insert(3);
        x.insert(4);

        y.insert(1);
        y.insert(2);


        result.insert(5);


        unite(x, y, result);
```

```
      assert(result.size() == 5 && result.contains(1) && result.contains(2) &&
result.contains(3) && result.contains(4) && result.contains(5));

      Set x;
      Set y;
      Set result;

      x.insert(1);
      x.insert(5);
      x.insert(3);
      x.insert(4);

      y.insert(1);
      y.insert(2);
      y.insert(3);
      y.insert(4);

      result.insert(5);
      result.insert(6);
      result.insert(7);
      result.insert(8);

      unite(x, y, result);
      assert(result.size() == 5 && result.contains(1) && result.contains(2) &&
result.contains(3) && result.contains(4) && result.contains(5));*/
}



Set ss; //default constructor
    ss.insert("lavash"); //testing insert
    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; //testing get
    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


Set ss; //default
//testing contains insert/erase and get()
    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 == "");
//testing swap
Set ss1;
    ss1.insert("bing");
    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("bing"));

//basic testing for string type
Set s;
        assert(s.empty());
        ItemType x = "arepa";
        assert( !s.get(42, x)  &&  x == "arepa"); // x unchanged by get
failure
        s.insert("chapati");
        assert(s.size() == 1);
        assert(s.get(0, x)  &&  x == "chapati");

Set ss;
            assert(ss.insert("roti"));
            assert(ss.insert("pita"));
            assert(ss.size() == 2);
            assert(ss.contains("pita"));
            ItemType x = "bing";
            assert(ss.get(0, x)  &&  x == "pita");
            assert(ss.get(1, x)  &&  x == "roti");


//basic testing for unsigned long

Set uls;
            assert(uls.insert(20));
            assert(uls.insert(10));
            assert(uls.size() == 2);
            assert(uls.contains(10));
            ItemType x = 30;
            assert(uls.get(0, x)  &&  x == 10);
            assert(uls.get(1, x)  &&  x == 20);


Set s;
        assert(s.empty());
        ItemType x = 9876543;
        assert( !s.get(42, x)  &&  x == 9876543); // x unchanged by get
failure
        s.insert(123456789);
        assert(s.size() == 1);
        assert(s.get(0, x)  &&  x == 123456789);
```

```
        cout << "Passed all tests" << endl;


//making sure get works properly
Set s;  // ItemType is int
        s.insert(20);
        s.insert(30);
        s.insert(10);
        int x = 999;
        assert(s.get(0, x)  &&  x == 10);
        assert(s.get(1, x)  &&  x == 20);
        assert(s.get(2, x)  &&  x == 30);
        cout << "All tests passed" << endl;
//testing subtract:

Set y;
       Set result;
//testing when s1 and s2 are the same set
       x.insert("a");
       x.insert("b");
       x.insert("c");
       x.insert("d");

       y.insert("a");
       y.insert("b");
       y.insert("c");
       y.insert("d");

       subtract(x, y, result);
       assert(result.empty()&&!x.empty()&&!y.empty());

       result.insert("x");
       result.insert("tiny");
       subtract(x, y, result);
       assert(result.empty() && !x.empty() && !y.empty());

Set x;
       Set y;
       Set result;

       x.insert("a");
       x.insert("b");
       x.insert("c");
       x.insert("d");

       y.insert("e");
       y.insert("f");
       y.insert("g");
       y.insert("h");

       subtract(x, y, result);
       assert(!result.empty() && result.contains("a") && result.contains("b") &&
result.contains("c") && result.contains("d"));

       result.insert("x");
       result.insert("tiny");
       subtract(x, y, result);
```

```
	assert(!result.empty() && result.contains("a") && result.contains("b") &&
result.contains("c") && result.contains("d"));

	//test cases for aliasing when s1 and result being the same
	subtract(x, y, x);
	subtract(y, x, y);
	assert(x.size() == 4 && x.contains("a") && x.contains("b") && x.contains("c") &&
x.contains("d"));
	assert(y.size() == 4 && y.contains("e") && y.contains("f") && y.contains("g") &&
y.contains("h"));
	x.insert("f");
	subtract(x, y, x);
	assert(x.size() == 4 && x.contains("a") && x.contains("b") && x.contains("c") &&
x.contains("d"));
	x.insert("f");
	subtract(y, x, y);
	assert(y.size() == 3 && y.contains("e") && y.contains("g") && y.contains("h"));

	//seperate test case for when aliasing with with s2 and result being the same
	//subtract(y, x, x);
	//assert(x.size() == 3 && x.contains("e") && x.contains("g") && x.contains("h"));
	subtract(x, x, x);
	assert(x.empty()); //x should be empty since its chnaged through result testing
for when s1,s2,and result are the same, input should be empty

//basic testing of subtract
Set x;
	Set y;
	Set z;

	x.insert("a");
	x.insert("b");
	x.insert("c");

	y.insert("coo");
	subtract(x, y, z);
	assert(z.size() == 3 && z.contains("a") && z.contains("b") && z.contains("c"));
//testing when result is empty and s1 and s2 are different with no duplicates
	subtract(y, x, z);
	assert(z.size() == 1 && z.contains("coo")); //testig when result has values and s1
and s2 are different with no duplicates

	y.insert("c");
	subtract(y, x, z);
	assert(z.size() == 1 &&
z.contains("coo")&&y.contains("coo")&&y.contains("c")&&y.size()==2
	&& x.size()==3 && x.contains("a") && x.contains("b") && x.contains("c")); //testig
when result has values and s1 and s2 are different with duplicates
	Set p;
	subtract(x, y, p); //testing when result is empty and s1 and s2 are different with
duplicates
	assert(p.size() == 2 && p.contains("a") && p.contains("b") && y.contains("coo") &&
y.contains("c") && y.size() == 2
		&& x.size() == 3 && x.contains("a") && x.contains("b") && x.contains("c"));
```