1. My Set class implements a doubly-linked list with a head and a tail pointer. My list nodes contain two Node pointers to the nodes who appear before and after the node, respectively. The nodes in the list are ordered from least to greatest, and the list itself is not circular. There are no dummy nodes implemented. Each node is ordered from least to greatest from left to right.
2. pseudocode:
   1. assignment operator

empty out calling set

traverse reference set

declare new nodes with corresponding values

update previous and next

return \*this

* 1. copy constructor

traverse reference set

declare new nodes

fill corresponding values in this

update previous and next

* 1. insert

if set contains value

return false

if set is empty

allocate node as head and tail and add value

if set only has one node and value is greater than head value

allocate new node after head node

assign the node to tail pointer

else if value is less than head value

allocate new node as head node

else

traverse set until the node before greater node than value

place new node in spot

if value is greater than every node in set

place node at end, mark it as tail

* 1. erase

if set doesn’t contain value

return false

if set size is 1

set head node to nullptr

delete node

if head node is the target

reassign head pointer to second node in set

delete target node

return

repeatedly:

traverse to the next node in set

if target value

rearrange next and previous nodes

delete target

return

if not

continue

* 1. void unite(const Set& s1, const Set&s2, Set& result)

if result is equal to s1

get value from s2

insert into result

return

if result is equal to s2

get value from s1

insert into result

return

if result is not empty

repeatedly for all elements in result

get value on top of linked list

erase node

exit loop

repeatedly: for all elements in s1

get value at *i*th node in s1

insert value into result

repeatedly: for all elements in s2

get value at *i*th node in s2

insert value into result

* 1. void subtract (const Set& 21, const Set&s2, Set& result)

if result is not empty

repeatedly for all elements in result

get value on top of linked list

erase node

exit loop

//fill result

repeatedly: for all elements in s1

get *i*th value in s1

if s2 does not contain *i*th value in s1

insert *i*th value in s1 into result

1. Test cases:

Case 1: alias ItemType is std::string

Set a; //default constructor

string s = "cheese";

assert(a.empty() && a.size() == 0); //private data initialized

assert(!a.get(0, s) && s == "cheese"); //get returns false on empty set and s is unchanged

assert(a.insert("apple")); //assert that "apple" is successfully inserted into a

assert(a.erase("apple")); //erase a set of one element

assert(a.empty() && a.size() == 0); //private data should be updated.

a.insert("apricot");

a.insert("banana");

a.insert("durian");

a.insert("coconut");

assert(a.contains("banana")); //test if contains

assert(!a.contains("mango")); //test if doesn't contain

assert(!a.insert("durian")); //insert returns false if duplicate

assert(!a.erase("mango")); //erase returns false if value isn't in set

Set b(a); //copy constructor

string t = "ham";

assert(a.get(3, s) && s == "durian"); //s is changed if pos is inbounds

assert(b.get(3, t) && t == s); //t has the same values as a

b.insert("grapefruit");

assert(!a.contains("grapefruit")); // a and b are independent sets

Set c;

c = b; //assignment operator

assert(c.get(4, s) && s == "grapefruit" && b.get(4, t) && t == s); //b and c have identical values

assert(c.erase("apricot") && b.contains("apricot")); //c and b are independent sets

a = a; //assignment operator on self

assert(a.contains("banana") && a.size() == 4); //a isn't deleted by accident

Set d;

b.swap(d);

assert(b.empty() && d.contains("grapefruit") && d.size() == 5); //sets switch data with one another

d = c;

d.insert("parrot");

assert(!d.contains("apricot")); //previous data gets deleted

Set e;

e.insert("avocado");

e.insert("broccoli");

unite(e, c, b); //unite two sets with different values

assert(b.contains("durian") && b.size() == c.size() + e.size());

Set f(c);

unite(f, c, f);

assert(f.get(2, s) && s == "durian" && f.size() == 4); //s1 is the same as result

subtract(b, e, d);

assert(!d.contains("parrot") && d.size() == c.size()); //previous data successfully wiped in subtract

Set z;

Set y;

Set x;

z.insert("a");

z.insert("b");

z.insert("c");

y.insert("1");

y.insert("2");

unite(z, z, x); //unite if s1 = s2

string p;

assert(x.get(0, p) && z.get(0, s) && s == p); //x should be identical to z

subtract(z, z, x); //subtract if s1 = s2

assert(x.empty()); //x should be empty

unite(x, y, x); //unite with result = s1

assert(x.get(1, p) && y.get(1, s) && p == s); //x should be identical to y

subtract(x, y, x); //s1 = result

assert(x.empty());//x should be empty

unite(z, y, x);

subtract(z, x, x);//s2 = result

assert(x.empty());

x.insert("almond");

x.insert("bean");

x.insert("carrot");

x.insert("duck");

subtract(x, x, x); //all three arguments are the same

assert(x.empty());

//Test 2, unsigned long

Set a;

assert(a.insert(100));

for (int i = 2; i < 6; i++) {

a.insert(i \* 100);

}

assert(a.erase(300));

assert(!a.erase(900));

Set b;

for (int i = -10; i < -1; i++) {

b.insert(i);

}

unsigned long s;

b.get(3, s);

assert(s == -7);

a.swap(b);

assert(a.contains(-5));

Set c;

a.swap(c);

assert(a.empty());

Set d(c);

//Test 2, unsigned long

Set a;

assert(a.insert(100));

for (int i = 2; i < 6; i++) {

a.insert(i \* 100);

}

assert(a.erase(300));

assert(!a.erase(900));

Set b;

for (int i = -10; i < 0; i++) {

b.insert(i);

}

unsigned long s;

b.get(3, s);

assert(s == -7);

a.swap(b);

assert(a.contains(-5));

Set c;

a.swap(c);

assert(a.empty());

Set d(c);

assert(d.contains(-3) && d.contains(-6));

assert(d.erase(-4));

Set e;

unite(b, d, e);

assert(e.size() == 13);

Set f;

f = e;

subtract(f, e, f);

assert(f.empty());