**a description of the design of your doubly-linked list implementation. (A couple of sentences will probably suffice, perhaps with a picture of a typical Set and an empty Set. Is the list circular? Does it have a dummy node? What's in your list nodes? Are they in any particular order?)**

I used circular doubly-linked list without a dummy node.

Things in my list nodes are the element inserted by member functions.

They are in an increasing order.

[**pseudocode**](http://web.cs.ucla.edu/classes/spring17/cs32/pseudocode.html) **for non-trivial algorithms (e.g., subtract).**

**Psudocode for unite function**

Create a temporary item as a medium to get the value from get function

If s1 and result is the same object

Repeatedly:

Assign the the item in s2 in certain position to temporary item

Insert the value of temporary item into s1

If s2 and result is the same object

Repeatedly:

Assign the item in s1 in certain position to temporary item

Insert the value of temporary item into s2

If no case above

Remove all item in the result Set

Repeatedly:

Assign the item in s1 in certain position to temporary item

Insert the value of temporary item into result Set

Repeatedly:

Assign the item in s2 in certain position to temporary item

Insert the value of temporary item into result Set

**Psudocode for subtract function**

Create a temporary item as a medium to get the value from get function

If s2 and result is the same

Create a new Set to be the same as s2

Repeatedly:

Assign the item in s1 in certain position to temporary item

Insert the value of temporary item into result Set

Remove all elements in s2 from the result Set

If s1 and result is the same

Repeatedly:

Assign the item in s2 in certain position to temporary item

remove the value of temporary item into result Set

if no such case above

remove all the item from result Set

repeatedly:

Assign the item in s1 in certain position to temporary item

Insert the value of temporary item into result Set

Repeatedly:

Assign the item in s2 in certain position to temporary item

remove the value of temporary item into result Set

**a list of test cases that would thoroughly test the functions. Be sure to indicate the purpose of the tests. For example, here's the beginning of a presentation in the form of code:**

#include "Set.h"

#include <iostream>

#include <cassert>

using namespace std;

void testCopy(Set t);

void test11();

void test22();

#include <type\_traits>

#define CHECKTYPE(f, t) { auto p = (t)(f); (void)p; }

static\_assert(std::is\_default\_constructible<Set>::value,

"Set must be default-constructible.");

static\_assert(std::is\_copy\_constructible<Set>::value,

"Set must be copy-constructible.");

void thisFunctionWillNeverBeCalled()

{

CHECKTYPE(&Set::operator=, Set& (Set::\*)(const ItemType&));

CHECKTYPE(&Set::empty, bool (Set::\*)() const);

CHECKTYPE(&Set::size, int (Set::\*)() const);

CHECKTYPE(&Set::insert, bool (Set::\*)(const ItemType&));

CHECKTYPE(&Set::erase, bool (Set::\*)(const ItemType&));

CHECKTYPE(&Set::contains, bool (Set::\*)(const ItemType&) const);

CHECKTYPE(&Set::get, bool (Set::\*)(int, ItemType&) const);

CHECKTYPE(&Set::swap, void (Set::\*)(Set&));

CHECKTYPE(unite, void (\*)(const Set&, const Set&, Set&));

CHECKTYPE(subtract, void (\*)(const Set&, const Set&, Set&));

}

int main()

{

Set test1;

test1.dump();

assert(!test1.contains(123)); //test contain when empty

assert(test1.empty()); //test empty

assert(test1.size() == 0); //test size when empty

assert(test1.insert(500)); //test insert

test1.dump();

assert(test1.contains(500)); //test contain

assert(test1.insert(999)); //insert at the end

assert(test1.insert(100)); //insert at the beginning

assert(!test1.contains(123)); //test contain when there is no such item

assert(test1.contains(100)); //test contain when not empty

assert(!test1.insert(100)); //insert an existing object

assert(!test1.empty()); //test empty when not empty

assert(test1.size() == 3); //test size when not empty

assert(test1.insert(109)); //in the middle

assert(!test1.insert(109)); //insert existing object

test1.dump();

assert(!test1.erase(923)); //test erase an item not in the list

assert(test1.erase(100)); //test erase at the beginning

test1.dump();

assert(test1.erase(999)); //test erase at the end

assert(!test1.erase(100));

assert(test1.insert(100));

test1.dump();

assert(test1.erase(109)); //test erase in the middle

assert(test1.size() == 2); //test size after erase

assert(test1.erase(100)); //

assert(test1.erase(500));

assert(test1.empty()); //test empty after erasing

test1.dump();

assert(test1.insert(100));

assert(test1.insert(500));

assert(test1.insert(999));

test1.dump();

//copy

Set test2;

assert(test2.insert(123));

assert(test2.insert(234));

assert(test2.insert(345));

assert(test2.insert(456));

assert(test2.insert(567));

assert(test2.insert(678));

Set pop = test1; //test copy-

test1 = test2; //test assign

test1.dump();

testCopy(test1); //-test assign operator and copy constructor

assert(test1.size() == 6); //test copy constructor that it does not change the original

cerr <<endl;

test1.dump();

ItemType x;

assert(test1.get(0,x) && x == 123); //test get when i = 0

assert(test1.get(5,x) && x == 678); //test get when i = size-1

assert(!test1.get(6,x)); //test get when i = size

assert(!test1.get(-1, x)); //test get when i < 0

//test11();

test22(); //test code from website

Set test3;

assert(test3.insert(124235));

assert(test3.insert(87324));

assert(test3.insert(2315));

unite(test1, pop, test3); //test unite with 3 different Set:

test3.dump();

assert(test3.size()==9);

assert(test3.contains(123));

assert(test3.contains(234));

assert(test3.contains(345));

assert(test3.contains(456));

assert(test3.contains(567));

assert(test3.contains(678));

assert(test3.contains(100));

assert(test3.contains(500));

assert(test3.contains(999));

Set jim;

subtract(test3, pop, jim); //test subtract with 3 different

jim.dump();

assert(jim.size()==6);

unite(test1, pop, test1); //test unite with s1 and result aliasing

assert(test1.size() == 9);

assert(test1.contains(123));

assert(test1.contains(234));

assert(test1.contains(678));

assert(test1.contains(500));

assert(test1.contains(100));

subtract(test1, pop, test1); //test subtract with s1 and result aliasing

assert(test1.size() == 6);

assert(test1.contains(123));

assert(test1.contains(234));

assert(test1.contains(345));

assert(test1.contains(456));

assert(test1.contains(567));

assert(test1.contains(678));

unite(test1, pop, pop); //test unite with s2 and result aliasing

assert(pop.size() == 9);

assert(pop.contains(123));

assert(pop.contains(567));

assert(pop.contains(500));

subtract(test1, pop, pop); //test subtract with s2 and result aliasing

assert(pop.empty());

unite(test1, test1, pop); //test unite with s1 and s2 aliasing

assert(pop.size() == 6);

assert(pop.contains(123));

assert(pop.contains(234));

assert(pop.contains(567));

assert(pop.contains(678));

subtract(test1, test1, pop); //test subtract with s1 and s2 aliasing

assert(pop.empty());

unite(test1, test1, test1); //test 3 same unite aliasing

assert(test1.size() == 6);

assert(test1.contains(123));

assert(test1.contains(234));

assert(test1.contains(345));

assert(test1.contains(456));

assert(test1.contains(567));

assert(test1.contains(678));

test1 = test1; //test assign operator aliasing

assert(test1.size() == 6);

assert(test1.contains(123));

assert(test1.contains(234));

assert(test1.contains(345));

assert(test1.contains(456));

assert(test1.contains(567));

assert(test1.contains(678));

subtract(test1, test1, test1); //test subtract with 3 same aliasing

test1.dump();

assert(test1.empty());

}

void testCopy(Set t)

{

std::cerr << "Set t is: " << std::endl;

t.dump();

t.erase(123);

t.erase(234);

t.erase(345);

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

}

/\*

void test11()

{

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");

}

\*/

void test22()

{

Set uls;

assert(uls.insert(10));

assert(uls.insert(20));

assert(uls.size() == 2);

assert(uls.contains(20));

ItemType x = 30;

assert(uls.get(0, x) && x == 10);

assert(uls.get(1, x) && x == 20);

}