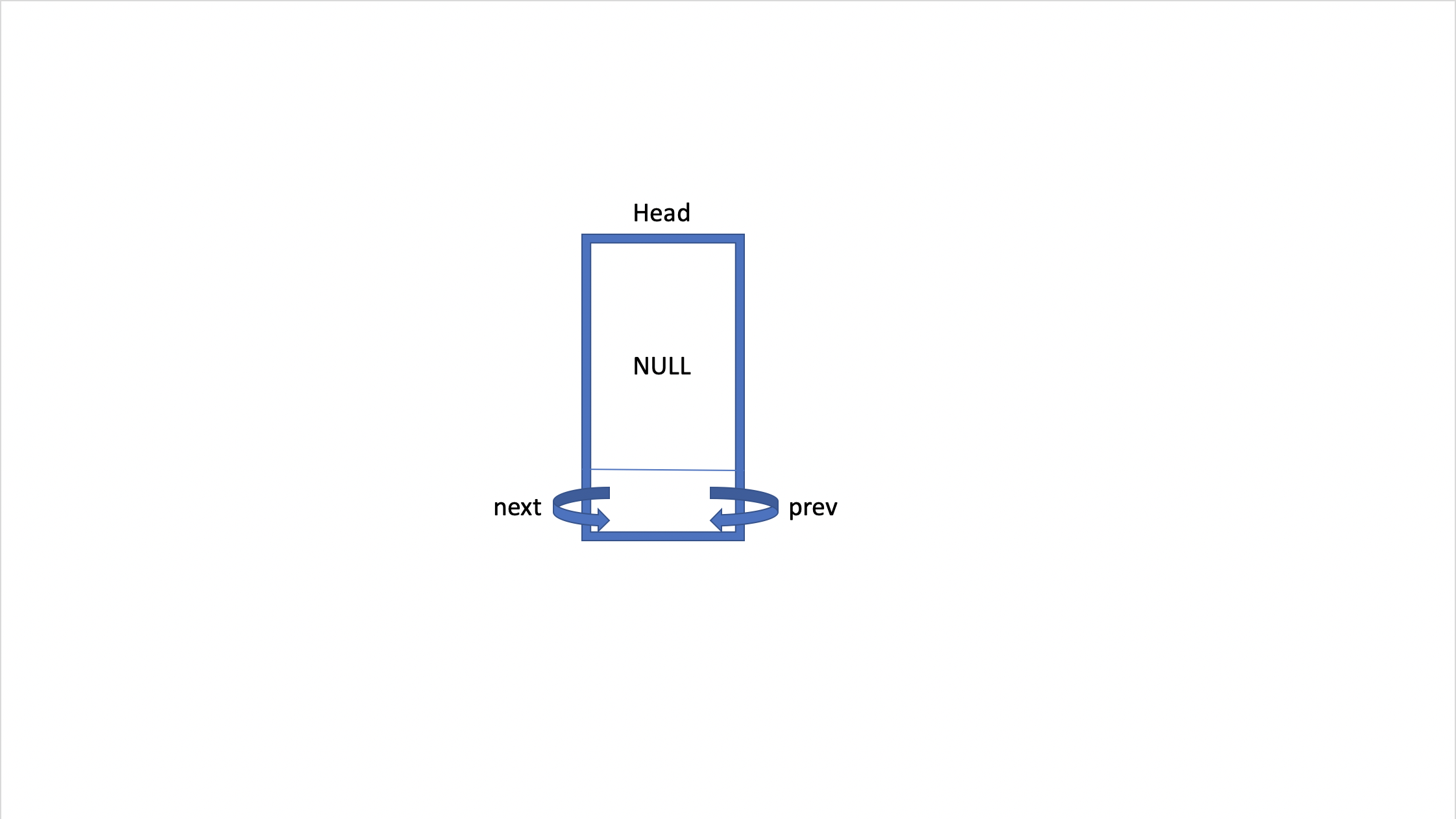
Nathan Tjoar

005081232

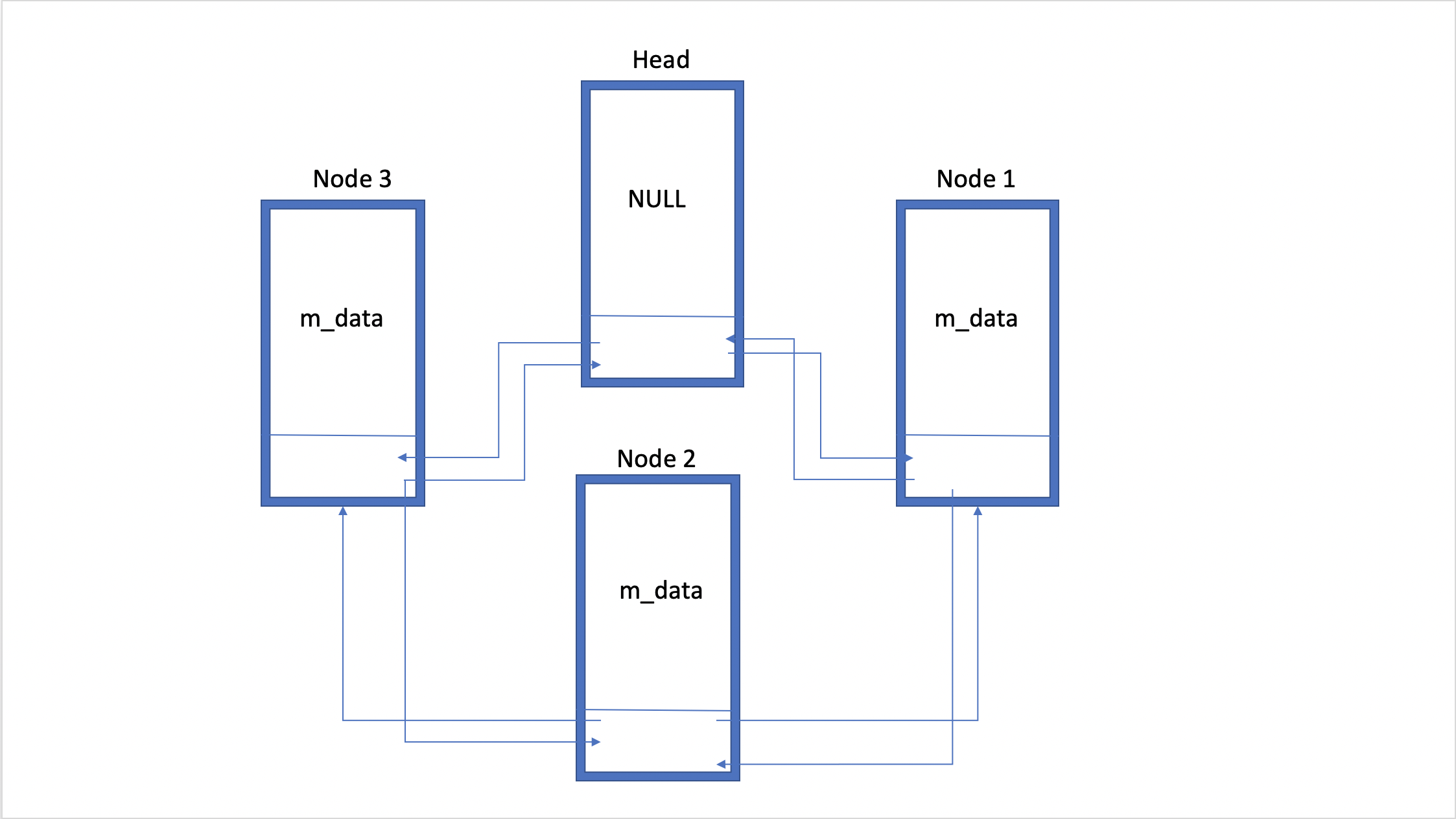
Report

1. The design of my project is a circular doubly linked list with a dummy variable being the head variable. The design was built so the “head” or dummy pointer could be called at any time to specifically be the main reference to the linked list. In my list nodes are simply a pointer to the next node, a pointer to the previous and a data type of type ItemType, which a type alias is able to handle.

The average empty list looks like the image shown below:



A linked list with some possible data as well is shown below:



1. Set.cpp pseudocode:

Default constructor

Set size to be zero

Make head point to itself

Destructor

Repeatedly (when the next node is not the head node)

Make temporary point to node after head

Set the temp’s next node’s prev to point to head

Set head’s next to point to temporary’s next

Delete temporary

When done, delete head

Assignment constructor

If the current object is not already the value object

Copy the value object to a temporary object

Swap the current object with temp

Return a pointer to this

Copy constructor

Create a new default node

Repeatedly

Create a new node

Allocate pointers to incorporate new node

Do so for the entire loop

Empty function

Return true if the list is empty

Size function

Return size of list

Insert function

If the value is not in the set,

Create a new node

Allocate pointers to properly incorporate new nodes

Return true

Else

Return false

Erase function

Set a temporary pointer pointing to the first value after head

If size is not zero

Repeatedly

Iterate through loop and search for given value

If found

Allocate pointers

Delete value

Return true

Else

Return false

Contains function

Set a temporary pointer pointing to the first value after head

Repeatedly

Iterate through loop and search for given value

If found

Allocate pointers

Return true

Else

Return false

Get function

If the pos is out of bounds

Return false

Create a rank to store how many values a specific data is greater than

For all values in the list (s1), repeatedly

Reset rank to zero

For all values in the list (s2), repeatedly

Compare the two values

If s1 is greater than s2

Increment rank

Search next item

If rank is equal to the given pos

Set value to equal s1’s item

Return true

Return false

Swap function

Set a temporary pointer pointing to other’s head

Set other’s head to this head

Set this head to other’s head

Store other’s size in a tempSize variable

Replace other’s size with this size

Replace this size with other’s size

Dump function

Repeatedly for all values not head

Print data inside

Unite function

Repeatedly for all values in s1

Get the value

Insert value to result

Repeatedly for all values in s2

Get the value

Insert the value into the result if it is not there yet

Subtract function

Repeatedly for all values in s1

Get the value

Insert value to result

Repeatedly for all values in s2

Get the value

Delete the value from the result if there is any

1. Test Cases:

String Tests:

#include "Set.h" //String

#include <iostream>

#include <cassert>

#include <type\_traits>

using namespace std;

#define CHECKTYPE(f, t) { auto p = static\_cast<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 Set&));

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&));

}

void test()

{

Set ss;

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

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

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

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

ItemType x = "laobing";

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

assert(ss.get(1, x) && x == "roti");

}

void test2()

{

Set s;

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

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

assert(!s.erase("K"));

assert(s.erase("roti"));

assert(!s.contains("K"));

assert(s.contains("pita"));

ItemType n = "HI";

assert(!s.get(1, n) && n == "HI");

}

void extra()

{

Set s;

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

assert(!s.insert("Cabbage"));

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

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

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

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

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

Set s2;

assert(s2.empty());

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

assert(!s2.insert("Cabbage"));

assert(s2.size() == 1);

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

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

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

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

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

assert(s2.size() == 6);

Set add;

Set sub;

unite(s, s2, add);

assert(add.contains("Cabbage"));

assert(add.contains("Tomato"));

assert(add.contains("Chicken"));

assert(add.contains("Orange"));

assert(add.contains("Apple"));

assert(add.contains("Cake"));

subtract(s, s2, sub);

assert(!sub.contains("Cabbage"));

assert(sub.contains("Tomato"));

assert(!sub.contains("Chicken"));

assert(!sub.contains("Orange"));

assert(!sub.contains("Apple"));

assert(!sub.contains("Cake"));

}

void sw()

{

Set s;

assert(s.insert("100"));

assert(!s.insert("100"));

assert(s.insert("5081232"));

assert(s.insert("911"));

assert(s.insert("314159265"));

assert(s.insert("27189234"));

assert(s.insert("88"));

Set s2;

assert(s2.insert("2000"));

assert(!s2.insert("2000"));

assert(s2.insert("42069"));

assert(s2.insert("911"));

assert(s2.insert("5081232"));

assert(s2.insert("100"));

assert(s2.insert("42"));

s.swap(s2);

assert(s.contains("5081232"));

assert(s.contains("2000"));

assert(s.contains("42069"));

assert(s2.contains("911"));

assert(s2.contains("88"));

assert(s2.contains("100"));

}

int main()

{

test();

test2();

extra();

sw();

Set ss; // ItemType is std::string

ss.insert("ccc");

ss.insert("aaa");

ss.insert("bbb");

ItemType x = "xxx";

assert(!ss.get(3, x) && x == "xxx"); // x is unchanged

assert(ss.get(1, x) && x == "bbb"); // "bbb" is greater than

// exactly 1 item

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

}

Unsigned long test cases:

#include "Set.h" //Unsigned longs

#include <iostream>

#include <cassert>

using namespace std;

void test()

{

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);

}

void test2()

{

Set s;

assert(s.insert(200));

assert(s.insert(1000));

assert(!s.erase(20));

assert(s.erase(200));

assert(!s.contains(999));

assert(s.contains(1000));

ItemType n = 10;

assert(!s.get(1, n) && n == 10);

}

void extra()

{

Set s;

assert(s.insert(100));

assert(!s.insert(100));

assert(s.insert(5081232));

assert(s.insert(911));

assert(s.insert(314159265));

assert(s.insert(27189234));

assert(s.insert(88));

Set s2;

assert(s2.insert(2000));

assert(!s2.insert(2000));

assert(s2.insert(42069));

assert(s2.insert(911));

assert(s2.insert(5081232));

assert(s2.insert(100));

assert(s2.insert(42));

Set add;

Set sub;

unite(s, s2, add);

assert(add.contains(5081232));

assert(add.contains(100));

assert(add.contains(88));

assert(add.contains(911));

assert(add.contains(42069));

assert(add.contains(42));

subtract(s, s2, sub);

assert(!sub.contains(5081232));

assert(sub.contains(88));

assert(!sub.contains(100));

assert(!sub.contains(911));

assert(!sub.contains(2000));

}

void sw()

{

Set s;

assert(s.insert(100));

assert(!s.insert(100));

assert(s.insert(5081232));

assert(s.insert(911));

assert(s.insert(314159265));

assert(s.insert(27189234));

assert(s.insert(88));

Set s2;

assert(s2.insert(2000));

assert(!s2.insert(2000));

assert(s2.insert(42069));

assert(s2.insert(911));

assert(s2.insert(5081232));

assert(s2.insert(100));

assert(s2.insert(42));

s.swap(s2);

assert(s.contains(5081232));

assert(s.contains(2000));

assert(s.contains(42069));

assert(s2.contains(911));

assert(s2.contains(88));

assert(s2.contains(100));

}

int main()

{

test();

test2();

extra();

sw();

Set result;

subtract(s1, s2, result);

result.dump();

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

}