REPORT

1. The Set is implemented using a circular Linked List. It starts with a head pointing to a dummy node which doesn’t hold any information, but only a pointers to either ends the ‘circle’. To traverse around the Linked List, a pointer points to the first element, and keeps moving forward until it reaches the original position again.
2. Pseudocodes:
3. Set::insert(const ItemType& value)

Check if the item already exists in the Set

If not:

Create a new Node item

Point its next pointer to head’s next pointer

Make the head’s next pointer point to it

Make the next item’s prev pointer to point to the new item

Point the new item’s prev pointer to head

return true;

if yes:

return false;

1. Set::erase(const ItemType& value)

Check if the item exists in the Set

If yes:

Create node pointer and traverse it until it reaches the element we need to delete

Reassign the next pointer of the previous Node to the next Node

Reassign the prev pointer of the next Node to the previous Node

Delete the element

Return true;

If not:

Return false;

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

Assign the value of s1 to result

Create a pointer that points to the head->m\_next of s2

While pointer!=s2.head

If s1 does not contain pointer->m\_data

Result.insert(pointer->m\_data)

1. Void subtract(const Set& s1, const Set& s2, Set& result)

Assign the value of s1 to result

Create a pointer that points to the head-<m\_next of s2

While pointer!=head

If s1 contains pointer->m\_data

Result.erase(pointer->m\_data)

1. TEST DATA:

#include<iostream>

#include"Set.h"

#include<cassert>

using namespace std;

int main()

{

Set t1;

assert(t1.empty()); //checks if the newly initialized Set is empty

assert(t1.insert(10)); //checks if insert function works

assert(!t1.empty()); //checks if after inserting, empty function returns false

assert(t1.size() == 1); //checks if size == 1

assert(t1.contains(10));//checks if the Set contains 10

assert(t1.insert(20));

assert(t1.size() == 2); //checks if size returns the correct value

assert(t1.contains(20));

Set t2;

assert(t2.insert(1));

assert(t2.insert(2));

assert(t2.insert(3)); //inserting three elements into the Set

assert(t2.contains(1));

assert(!t2.contains(4));//checks if it returns false when asked if the set contains 4

assert(t2.size() == 3);

cout << "T1\n";

t1.dump();

cout<<"T2\n";

t2.dump(); //Before Swapping

t1.swap(t2);

assert(t1.size() == 3);

assert(t2.size() == 2);

cout << "T1\n";

t1.dump();

cout << "T2\n";

t2.dump(); //After Swapping

Set t3;

unite(t1, t2, t3);

assert(t3.size() == 5);

assert(t3.contains(1));

assert(t3.contains(20));

cout << "T3:\n";

t3.dump(); //Testing the unite function

Set t4;

t4 = t3; //Checking if operator= function works

assert(t4.size() == 5);

assert(t4.contains(1));

assert(t4.contains(20));

cout << "T4:\n";

t4.dump();

Set s1;

assert(s1.insert(2));

assert(s1.insert(4));

assert(s1.insert(6));

assert(s1.insert(8));

Set s2;

assert(s2.insert(6));

assert(s2.insert(8));

assert(s2.insert(10));

assert(s2.insert(12));

Set s3;

subtract(s1, s2, s3);

///NOTE: s3 must contain only 2 and 4

assert(s3.size() == 2);

assert(s3.contains(2));

assert(s3.contains(4));

Set q1;

assert(q1.insert(11));

assert(q1.insert(12));

assert(q1.insert(13));

assert(q1.size() == 3);

assert(!q1.erase(10));//Checking what happens if we try to erase an item that is not present

assert(q1.size() == 3);//...nothing...

assert(q1.erase(11));

assert(q1.erase(12));

assert(q1.size() == 1);//checking if the size function returns the right value

assert(!q1.empty());

assert(q1.erase(13));

assert(q1.empty());

cout << "passed all tessts" << endl;