**Description**

This project uses a doubly linked list with a head and a tail node. Each list node contains some data of ItemType, along with pointers to the previous and next node. The previous node pointer of the head node and the next node pointer of the tail node are null. Each node contains a value that is lexicographically greater than the next, and this is decided when the element is inserted into the set.

| Empty Set | Typical Set |
| --- | --- |
| Set Address: 2000  Head Address: NULL  Tail Address: NULL  m\_size: 0 | Set Address: 5000  Head Address: 5000  Tail Address: 9789  m\_size: 3  *Element 1:*  Address: 5000  m\_prev: NULL  m\_next: 6154  *Element 2:*  Address: 6154  m\_prev: 5000  m\_next: 9789  *Element 3:*  Address: 9789  m\_prev: 6154  m\_next: NULL |

**Pseudocode**

contains:

while element in list

allocate a new node to first element

allocate a new node to last element

if value of either node is value,

return true

update position of first node

update position of second node

return false

get:

if position is within the range of list,

allocate a new node to the first element

repeatedly:

move to next element until counter reaches position

copy node’s value, return true

return false if position not in range

erase:

if list doesn’t contain value or empty list,

return false

if value to be deleted is first element,

killMe = address of top node

update head to point to the second node in the list

delete target node

update set size

return true

allocate a new node to first element

traverse every element in the set

if next element’s data matches value,

break

move to next element

if value has been found,

killMe = address of new node

link the node above to the node below

delete target node

update set size

return true

insert:

if list is empty

allocate a new node with value and no elements before and after it

set first element and last element of list to new node

update set size

return true

if list doesn’t already contain value,

if value should be added as first element to list,

allocate new node with value

link the new node above the head node

set head to first element

update set size

return true

if value should be added as last element to list,

allocate new node with value

link the new node below the tail node

set tail to last element

update set size

return true

if value should be placed somewhere in the middle of the list,

allocate a new node pointing to first element

keep traversing list until reach where value should be placed

allocate new node with value

link the new node to the appropriate element before and after it

update size

return true

return false

swap:

temporarily store the new set’s size

set new set’s size to old set’s size

set old set’s size to temporarily stored size

temporarily store the address to new set’s first element

set address to new set’s first element to address to old set’s first element

set address to old set’s first element to temporarily stored address

temporarily store the address to new set’s last element

set address to new set’s last element to address to old set’s last element

set address to old set’s last element to temporarily stored address

unite:

repeatedly until end of first set,

use insert on each element into result set

repeatedly until end of second set,

use insert on each element into result set

butNot:

repeatedly until end of first set,

get each element

if second set doesn’t contain element,

use insert on that element into result set

Copy Constructor:

if the old set is empty,

then set the new set as empty

return

create a new head node and set to value of old head node

set head’s previous and next elements that are pointed to as empty

move to the second element of the old list

repeatedly until the end of the old list,

create a new node

set its values, and next and previous elements to the corresponding ones in the old list

set the tail node to the last element

set the new set size to the old set size

Default Constructor:

Initialize set size to 0

Make the head and tail nodes not point to anything

Set Destructor:

if set is empty

return

traverse through each element in set

delete node

move to next node

Assignment Operator:

if the new set is not the same as the current set,

create new copy of rhs using copy constructor

swap elements of the two sets

return the set itself

**Test Cases**

// default constructor

Set ss;

// For an empty set:

assert(ss.size() == 0); // test size

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

assert(!ss.erase("e")); // nothing to remove

ss.insert("to be erased");

ss.erase("to be erased"); //testing erase at first element

assert(ss.size() == 0); //checking if erase resulted in the empty set

ss.insert("e"); //testing insert

ss.insert("a");

ss.insert("c");

ss.insert("d");

ss.insert("to be erased");

ss.erase("to be erased"); //testing erase in the middle

assert(ss.size() == 4);

ss.insert("a"); //trying to insert duplicates

ss.insert("b");

ss.insert("d"); //trying to insert duplicate at end

ss.insert("to be erased");

ss.erase("to be erased"); //testing erase at last element

assert(ss.size() == 5); // duplicate "a" and "d" were not added

assert(ss.contains("a")); //test contains

assert(!ss.contains("x")); //test not containing

assert(!ss.contains("")); //test empty string

string x;

ss.get(0, x); //test get at first element

assert(x == "a"); // "a" is greater than exactly 0 items in ss

ss.get(4, x); //test get at last element

assert(x == "e"); // "e" is greater than exactly 4 items in ss

ss.get(2, x); //test get at middle lement

assert(x == "c"); // "c" is greater than exactly 2 items in ss

Set ss1;

ss1.insert("x");

Set ss2;

ss2.insert("y");

ss2.insert("z");

ss1.swap(ss2); //test swap

assert(ss1.size() == 2 && ss1.contains("y") && ss1.contains("z") &&

ss2.size() == 1 && ss2.contains("x"));

Set ss3 = ss1; //test copy constructor

Set ss4;

ss4 = ss2; //test assignment operator

Set ss5; //testing strings with multiple characters

ss5.insert("ccc");

ss5.insert("aaa");

ss5.insert("bbb");

ItemType y = "xxx";

assert(!ss5.get(3, y) && y == "xxx"); // x is unchanged

assert(ss5.get(1, y) && y == "bbb"); // "bbb" is greater than

// exactly 1 item

Set ss6;

ss6.insert("x");

ss6.insert("y");

ss6.insert("z");

ss6.insert("a"); //add "a": common element between ss and ss6

Set result;

unite(ss,ss6,result); //test unite: should print out abcdexyza

Set result2;

butNot(ss,ss6,result2); //test butNot: should print out bcde

Set ss8; Set ss9;

ss8.insert(""); //checking empty string

ss9.insert("a");

ss8.swap(ss9); //testing swap on empty string

Set ss10 = ss9; //testing copy constructor on empty string

Set ss11;

ss11 = ss9; //testing assignment operator on empty string

//Basic Testing on Unsigned Longs

// 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);