**Description**

This doubly-linked list is implemented by using a class that has public functions to perform changes to the list and output information about it such as the size and whether the list is empty or not. The linked list is made by nodes, which are implemented as private functions. These nodes include the data that the nodes carry, a pointer to the next node, and a pointer to the previous node. The linked list only utilizes a head pointer, is not circular, and does not use a dummy node. The linked list is created by adding new nodes to the end of the list.

**Pseudocode**

Empty function:

If head pointer points to nothing,

Return true, the list is empty

If it does point to a node,

Return false, the list is not empty

Size function:

If empty,

Return 0

Traverse list until the end,

Count 1 for every node passed

Return that count

Insert function(value):

If list contains value,

Return false

If list is empty,

Create new node

Make head pointer point to that node

Return true

If list is not empty,

Make new node

Traverse list to the end node

Link prev and next pointers of node to link with the end node

Return true

Erase function(value):

If list does not contain value

Return false

If the list size == 1,

Make head pointer point to nothing

Delete the node

Return true

If the value is the node that the head points to,

Make head pointer point to the next node in list

Make the node that the head now points to have the prev pointer point to nothing

Delete the node containing value

Return true

Else,

Find the node that contains value

Link the previous node and the next node together

Delete the node that contains the value

Return true

Contains function(value):

Traverse list until the node with the desired value is found

If the node with value is found

Return true

If not, keep going through the linked list

Return false if the value is not found in any node

Get function(position, variable):

If the desired position is greater than size of the list or less than 0,

Return false

Make a temporary copy of the linked list, and using that copy:

If the size of list is only 2,

Arrange the list so that the values are in order

If the size of list is greater than 2,

Repeatedly,

While the next node exists,

If the current value is greater than the next value

If the current value is in the head node

Arrange list so that it is in order

If the node after the next node does not exist

Arrange the list so that it is in order

Otherwise,

Arrange the list so that it is in order

If the current node’s next pointer points to nothing

Continue

Otherwise, move on to next node

If the desired position == 0,

Store the the head value in the given variable

Otherwise, traverse the arranged list until the desired position is reached

Store that position’s value into the variable

Return true

Swap function(other):

Create copy of head pointer of this set

Change this’ head pointer to other’s head pointer

Change other’s head pointer to the copy of this’ head pointer

Unite function(set 1, set 2, result set):

If result set is not empty,

repeatedly,

Get the value of each node

Erase each node

Repeatedly,

Get each value from set 1

If result set does not contain that value

Insert the value into result

Repeatedly,

Get each value from set 2

If result set does not contain that value

Insert the value into result

Subtract function(set 1, set 2, result set):

If result set is not empty,

Repeatedly,

Get value of each node

Erase each node

Repeatedly,

Get each value from set 1

If set 2 does not contain the value

Insert the value into result set

**Test Cases**

(using ItemType = string)

#include "Set.h"

#include <iostream>

#include <cassert>

using namespace std;

void test()

{

Set ss;

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

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

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

assert(ss.size() == 1); //test size when not empty

assert(!ss.empty()); //test for not empty

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

assert(ss.contains("pita")); //test contain

assert(!ss.contains("hello")); //test contain

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

assert(ss.erase("hi")); //test erase

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

assert(!ss.contains("hi"));

ItemType x = "laobing";

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

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

Set aa;

aa.insert("chicken");

aa.insert("toy");

aa.insert("apple");

aa.swap(ss);

assert(aa.contains("pita") && aa.contains("roti") && !ss.contains("roti") && ss.contains("chicken") && ss.contains("toy") && ss.contains("apple")); //test swap

}

int main()

{

test();

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

}

(using ItemType == unsigned long)

#include "Set.h"

#include <iostream>

#include <cassert>

using namespace std;

void test()

{

Set a;

a.insert(2);

a.insert(8);

a.insert(3);

a.insert(9);

a.insert(5);

Set b;

b.insert(6);

b.insert(3);

b.insert(8);

b.insert(5);

b.insert(10);

Set c;

c.insert(1);

Set d;

d.insert(5);

subtract(a, b, c);

unite(a, b, d);

assert(c.contains(2) && c.contains(9) && d.contains(9) && d.contains(3) && d.contains(6) && d.contains(5) && d.contains(10) && d.contains(2) && d.contains(8)); //test subtract and unite

}

int main()

{

test();

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

}