Jiunn Siow

Report

1. My linked lists are circular in design, and each list node includes a value and a pointer to the proceeding node. The elements used are value, next, and prev. The head and tail nodes each have a next and prev pointer.
2. void unite(const Set& s1, const Set& s2, Set& result)

* create int variable that will combine sizes of s1 and s2
* declare ItemType variables
* declare Set variable
* create for loop to traverse though list
* divide list into two. First list checks elements from 0 to size of s1. Second list checks elements starting after s1 to the end of s2.
* Use get and insert functions for first list check.
* Utilize get and insert functions for second list check, but also check for duplicates before implementing the insert function
* Utilize swap function to put elements of list j into our list

1. The tests were performed on a set of strings.

Set ss;

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

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

assert(ss.size() == 2); //tested set of size 2

assert(!ss.insert("jack"));

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

assert(ss.size() == 3); //test set of size 3

ItemType y = "trump";

//To test the insert and erase functions and see if other elements are affected

ss.get(0, y);

cout << y << endl;

assert(ss.get(1, y) && y == "pita");

assert(ss.erase("naan"));

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

//check contains function

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

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

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

ItemType x = "bing";

//check get function

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

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

Set mySet;

assert(mySet.insert("obama"));

assert(mySet.insert("putin"));

assert(mySet.insert("biden"));

assert(mySet.insert("assad"));

assert(mySet.insert("kim"));

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

assert(mySet.erase("putin"));

assert(mySet.erase("biden"));

assert(!mySet.contains("putin"));

//check after removal or insertions if any of the elements are still in Set

assert(!mySet.contains("biden"));

assert(mySet.contains("obama"));

assert(mySet.contains("assad"));

assert(mySet.contains("kim"));

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

Set ss1;

ss1.insert("bernie");

ss1.insert("hillary");

ss1.insert("obama");

Set ss2;

ss2.insert("trump");

ss2.insert("ted cruz");

Set ss3;

assert(ss3.insert("ucla"));

assert(ss3.size() == 1);

assert(ss3.erase("ucla"));

assert(ss3.size() == 0);

assert(ss3.empty());

//check swap function

ss1.swap(ss2);

assert(ss1.size() == 2 && ss1.contains("trump") && ss1.contains("ted cruz") && //make sure after the swap, each set contains the correct respective size and elements

ss2.size() == 3 && ss2.contains("bernie") && ss2.contains("hillary") && ss2.contains("obama"));

Set ss4;

ss4.insert("a");

ss4.insert("b");

ss4.insert("c");

ss4.insert("d");

ss4.insert("e");

Set ss5;

ss5.insert("c");

ss5.insert("d");

ss5.insert("e");

ss5.insert("f");

ss5.insert("g");

ss5.insert("h");

ss5.insert("i");

Set ss6;

ss6.insert("chinggis");

//check unite function

unite(ss4, ss5, ss6);

assert(ss6.contains("a"));

assert(ss6.contains("b"));

assert(ss6.contains("c"));

assert(ss6.contains("d"));

assert(ss6.contains("e"));

assert(ss6.contains("f"));

assert(ss6.contains("g"));

assert(ss6.contains("h"));

assert(ss6.contains("i"));

assert(!ss6.contains("chinggis"));

assert(ss6.size() == 9);

//check subtract function

Set ss7;

ss7.insert("ethan");

ss7.insert("leo");

ss7.insert("josh");

ss7.insert("jacob");

ss7.insert("bert");

ss7.insert("jack");

ss7.insert("jerry");

Set ss8;

ss8.insert("jon");

ss8.insert("hannah");

ss8.insert("obama");

ss8.insert("bert");

ss8.insert("jerry");

ss8.insert("joel");

ss8.insert("jack");

Set ss12;

ss9.insert("cameron");

ss9.insert("matt");

ItemType b;

subtract(ss7, ss8, ss9);

assert(ss9.contains("ethan"));

assert(ss9.contains("leo"));

assert(ss9.contains("josh"));

assert(ss9.contains("jacob"));

assert(!ss9.contains("cameron"));

assert(!ss9.contains("matt"));

assert(!ss9.contains("jon"));

assert(!ss9.contains("hannah"));

assert(!ss9.contains("obama"));

assert(!ss12.contains("bert"));

assert(!ss9.contains("jerry"));

assert(!ss9.contains("jack"));

assert(!ss9.contains("joel"));

cout << ss9.size() << endl;

assert(ss9.size() == 4);