David Lane 004638553

CS 32 Project 2 report

1. I used a non circular doubly linked list with a head and a tail pointer pointing to the first and last items in the list respectively for the set implementation. I did not use a dummy node which led me to require some workarounds (such as the one mentioned in the erase function comments). I dynamically create a node and then dynamically create new nodes and add them to the end of the list with each new item to be added to the set. These nodes are structs featuring a data member and a pointer to the previous and next item in the list. These are used to navigate and alter the list. Besides the head and the tail pointers I also have a ptr Node pointer which is used in several functions to help with adjusting the pointers within the nodes as necessary.
2. a) Default constructor:

allocates a new node,

sets all pointers to point to it

sets the number of items in the set to 0

b) Copy Constructor:

allocates a new node

if set to be copied is not empty

copy it’s head data into set’s head.

Starting at the second item in the list

(if it exists)

Go through n,

allocate new nodes in the set for each of n’s nodes

and copy n’s data into those nodes,

keeping the newly constructed sets nodes linked by adjusting its pointers

c) operator=:

if the two sets aren’t at the same address

copy construct a temporary set j equivalent to n

use swap to set this to j

return this

d) insert:

if nothings in the list yet

set data in the first node to value

return

otherwise go through the list until either encountering the end or data the same as value

if at the end then add a new node

to point to tail of the list

be pointed to by the list and

add value as its data

return true

if we found the value already in the list return false and do nothing

e) erase:

if the head is the value were looking for

if theres only one item

delete it and then recreate it now with no data

reduce numitems

if there are more than one item in the list

set ptr to head

move head to the second item

delete ptr

reduce numitems

if the value is at tail

same thing as for head

otherwise use for loop to go through the list

same thing as for insert only this time

if we find the value

we delete that node

and return true

if we don’t we return false

f) get:

i needs to be within the domain of possible number of data members value could be larger than, if so

go through each node in the list

compare to each other item in the set

if the number of nodes’ data that nodes data is greater than is i

add that data to value

g) swap:

create a temporary in and pointer

use these to store and switch each private member of two sets’s value

h) unite:

create a temporary set

add all members of s1 to that set

add all members of s2 to that set that aren’t already in it

set result to the temp set

delete the temp set

1. subtract:

create a temporary set

add all members of s1 to that set

erase all members of s2 that are in s1 from temp

set result to the temp set

delete the temp set

3.The tests were performed on a set of strings (i.e., the ItemType typedef specified std::string).

// default constructor

Set s; //create new set

assert(s.size() == 0); //ensure size is 0

assert(s.empty()); //ensure set is empty

assert(s.insert("david")); //add david

assert(!s.insert("david")); //cant add dup

assert(s.size() == 1);//check size works if something is present

Set me(s); //tests for copy contructor with one item

assert(!me.insert("david"));

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

assert(s.erase("david"));//tests erase

assert(s.insert("david"));//these two are to test a bug where deleting with one item and inserting again did not work

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

assert(me.size() > s.size());

assert(me.contains("david") && !s.contains("david")); //tests contains

assert(me.insert("smallberg"));

string x;

assert(me.get(1, x) && x == "smallberg");//tests get and creates multiple nodes

assert(me.insert("stahl"));

assert(me.get(2, x) && x == "stahl");

assert(me.erase("stahl"));

assert(!me.contains("stahl"));

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

s.swap(me); //tests swap function

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

assert(!me.contains("smallberg"));

me = s; //tests for assignment operator

assert(me.get(1, x) && x == "smallberg");

assert(s.get(1, x) && x == "smallberg");

assert(!s.empty()); //ensures empty return false when not empty

Set j(me); //tests copy constructor with multiple nodes

assert(j.get(1, x) && x == "smallberg");

cout << "yay! pray for no invisible memory leaks" << endl;