CS32

Project 2

Cynthia Nguyen

Design implementation:

I created a doubly-linked list, each node consisting of three values: first name of string, last name of string, value of GuestType, an address to the previous node, and an address to the next node. The head’s previous address and the tail’s next address both are nullptrs. An empty list would have both head and tail be nullptrs. I chose this so that I could know that the list reaches it’s end when it encounters a nullptr.

Notable Obstacles:

* inviteGuest did not check the case that the guest comes after the guest at the tail, solved with another if statement
* The prev would not correctly contain the address of the previous node, was an indexing issue, solved in inviteGuest
* Alphabetical sorting was not working correctly, solution: was breaking out of loop at incorrect node, iterate to node before guest is inserted in inviteGuest
* verifyGuestOnTheList if statement forgot to be changed from (0<=i<guestCount()) to ( 0<= i && i <guestCount() ), so when guestCount() == 1 and i == 0, verifyGuestOnTheList would return false

PseudoCode:

* Copy Constructor:

Return if list is empty

Create head node of copy

Copy all values of rhs head into copy head

Iterate to next node

Loop for every node

{

Create new node

set copy’s next to address of new node

Iterate copy to new node

Set prev address to previous node

Copy all values of rhs’s node into new node

Make the new node the tail

}

* Assignment operator:

Return \*this if list is empty or if assigning to the same WeddingGuest list

Delete all nodes in this

Copy all nodes of rhs into this just like with copy constructor, (copy and paste partial copy constructor)

Return \*this

* inviteGuest:

return false if name is already in list

insert to front if list is empty

insert to front if name comes first alphabetically

insert to rear if name comes last alphabetically

find where name comes in alphabetically

go to node above where guest goes in

create new node for guest

copy all values into new node

set address for node above, below, and for new node

* InviteOrAlter:

If name is found, alter value

Else, do the same as inviteGuest() except for when name is in list

* crossGuestOff:

return false if list is empty

if guest is head, set head as next node, new head’s prev addressed to nullptr, delete head

if guest if tail, set tail as previous node, new tail’s next addresses to nullptr, delete tail

if guest is in center, set addresses of node above and below the node to be crossed off to point to each other, delete guest

if guest is not on list, return false

* joinGuests:

create WeddingGuest joined

invite all guests on odOne to joined

invite guests on odTwo

if guest on odTwo is already invited on joined

if values on joined and odTwo of same guest are different, cross of guest, make sure function return false after rest of guests are invited

copy joined into odJoined

* attestGuests:

create WeddingGuest Result

if fsearch and lsearch are both “\*”, Result = odOne

if only fsearch is “\*”, copy all guests with lsearch last name into Result

if only lsearch is “\*”, copy all guests with fsearch first name into Result

else, copy guest with fsearch and lsearch names into Result

copy Result into odResult

Test cases:

All tests are the same for each GuestType.

void testdouble() {

//all tets run with GuestType == double

// default constructor

WeddingGuest lal;

// For an empty list:

assert(lal.guestCount() == 0); // test size

assert(lal.noGuests()); // test empty

assert(!lal.crossGuestOff("Malik", "Monk")); // nothing to erase

WeddingGuest youngins;

assert(youngins.inviteGuest("Lauren", "U", 23));//tests invite guest

assert(youngins.inviteGuest("James", "H", 29));

assert(youngins.guestCount() == 2);//tests guest count

string first, last;

double a;

assert(youngins.verifyGuestOnTheList(0, first, last, a) && a//tests verifyGuestsOnTheList of first index

== 29);

assert(youngins.verifyGuestOnTheList(1, first, last, a) &&//tests verifyGuestsOnList of last index

(first == "Lauren" && a == 23));

WeddingGuest groomsmen;

groomsmen.inviteGuest("Tony", "Ambrosio", 40);

groomsmen.inviteGuest("Long", "Le", 41);

groomsmen.inviteGuest("Mike", "Wu", 43);

groomsmen.inviteGuest("Robert", "Wells", 44);

groomsmen.inviteGuest("Justin", "Sandobal", 37);

groomsmen.inviteGuest("Nelson", "Villaluz", 38);

WeddingGuest bridesmaids;

bridesmaids.inviteGuest("Serra", "Park", 39.5);

bridesmaids.inviteGuest("Saadia", "Parker", 37.5);

assert(!bridesmaids.invitedToTheWedding("", ""));//tests invitedToWedding when guest isnt invited

bridesmaids.inviteGuest("Patricia", "Kim", 39.0);

bridesmaids.inviteGuest("", "", 21.0);

bridesmaids.inviteGuest("Kristin", "Livingston", 38.0);

assert(bridesmaids.invitedToTheWedding("", ""));//tests invitedtoWedding when guest is invited

bridesmaids.crossGuestOff("Patricia", "Kim");

assert(bridesmaids.guestCount() == 4

&& bridesmaids.invitedToTheWedding("Serra", "Park")//tests invitedToWedding, checks all invted guests are on list

&& bridesmaids.invitedToTheWedding("Saadia", "Parker")

&& bridesmaids.invitedToTheWedding("Kristin", "Livingston")

&& bridesmaids.invitedToTheWedding("", ""));

//checks alphabetical order

//groomsmen.dump();

//bridesmaids.dump();

//test crossGuestOff

groomsmen.crossGuestOff("Robert", "Wells");

groomsmen.crossGuestOff("Mike", "Wu");

assert(!groomsmen.crossGuestOff("Cynthia", "Nguyen"));//tests when guest is not on list

assert(groomsmen.guestCount() == 4);

//tests cross all guests off

WeddingGuest test0;

GuestType value;

test0.inviteGuest("test", "test", 0);

test0.inviteGuest("test1", "test1", 0);

test0.inviteGuest("test2", "test2", 0);

while(!test0.noGuests())//tests noGuests

{

test0.verifyGuestOnTheList(0, first, last, value);

test0.crossGuestOff(first, last);

}

assert(test0.noGuests());//tests noGuests

//cout << "pass test 0" << endl;

//tests swapWeddingGuests

groomsmen.swapWeddingGuests(bridesmaids);

assert(groomsmen.guestCount() == 4

&& groomsmen.invitedToTheWedding("Serra", "Park")

&& groomsmen.invitedToTheWedding("Saadia", "Parker")

&& groomsmen.invitedToTheWedding("Kristin", "Livingston")

&& groomsmen.invitedToTheWedding("", ""));//test that groomsmen contains bridesmaids

//groomsmen.dump();

//bridesmaids.dump();

bridesmaids.swapWeddingGuests(groomsmen);

assert(bridesmaids.guestCount() == 4

&& bridesmaids.invitedToTheWedding("Serra", "Park")

&& bridesmaids.invitedToTheWedding("Saadia", "Parker")

&& bridesmaids.invitedToTheWedding("Kristin", "Livingston")

&& bridesmaids.invitedToTheWedding("", ""));//tests that bridesmaids contains bridesmaids again

//tests copy constructor, assignment operator

string first1;

string last1;

GuestType value1;

WeddingGuest temp = groomsmen;//cpy constructor

assert(groomsmen.verifyGuestOnTheList(0, first, last, value) == temp.verifyGuestOnTheList(0, first1, last1, value1) && groomsmen.guestCount() == temp.guestCount());

groomsmen = bridesmaids;//assignment operator

assert(groomsmen.verifyGuestOnTheList(0, first, last, value) == bridesmaids.verifyGuestOnTheList(0, first1, last1, value1) && groomsmen.guestCount() == bridesmaids.guestCount());

groomsmen = temp;

cout << "pass constructor and assignment operator" << endl;

bridesmaids.inviteGuest("Tony", "Ambrosio", 0);

//tests matchInvitedGuest && alterGuest

assert(bridesmaids.alterGuest("Tony", "Ambrosio", 43) && bridesmaids.matchInvitedGuest("Tony", "Ambrosio", value) && value == 43);

assert(!bridesmaids.alterGuest("Cynthia", "Nguyen", 20)); //tests when guest not on list

//test inviteOrAlter

assert(bridesmaids.inviteOrAlter("Cynthia", "Nguyen", 19) && bridesmaids.guestCount() == 6); //test invite when not on list

assert(bridesmaids.inviteOrAlter("Tony", "Ambrosio", 60) && bridesmaids.matchInvitedGuest("Tony", "Ambrosio", value) && value == 60 && bridesmaids.guestCount() == 6); //test alter when on list

//tests nonmember functions//

//tests joinGuests

WeddingGuest test1;

joinGuests(groomsmen, bridesmaids, test1);//odJined is empty

joinGuests(groomsmen, bridesmaids, temp);//odJoined containes values

assert(test1.verifyGuestOnTheList(0, first, last, value) == temp.verifyGuestOnTheList(0, first1, last1, value1) && test1.guestCount() == temp.guestCount());

attestGuests("\*", "\*", test1, temp);//both first and last are wild

//temp.dump();

WeddingGuest test2;

WeddingGuest test2\_1;

test2.inviteGuest("Anthony", "Nguyen", 18);

test2.inviteGuest("Cynthia", "Nguyen", 19);

test2.inviteGuest("Phuong", "Tran", 76);

test2.inviteGuest("Khanh", "Nguyen", 64);

test2.inviteGuest("Kelvin", "Thoi", 23);

test2.inviteGuest("Cynthia", "Tran", 19);

attestGuests("\*", "Nguyen", test2, temp);//test only first is wild and odResult is not empty

attestGuests("\*", "Nguyen", test2, test2\_1);//test only first is wild and odResult is empty

//temp.dump()

//test2\_1.dump()

assert(temp.guestCount() == test2\_1.guestCount());

assert(test2\_1.guestCount() == 3);

attestGuests("Cynthia", "\*", test2, temp);//test only last is wild

assert(temp.guestCount() == 2);

attestGuests("Cynthia", "\*", test2, test2);//tests if both odOne and odResults are same linked list

assert(test2.guestCount() == 2 && test2.invitedToTheWedding("Cynthia", "Nguyen") && test2.invitedToTheWedding("Cynthia", "Tran"));

return;

}

void testint() {

//all tets run with GuestType == int

// default constructor

WeddingGuest lal;

// For an empty list:

assert(lal.guestCount() == 0); // test size

assert(lal.noGuests()); // test empty

assert(!lal.crossGuestOff("Malik", "Monk")); // nothing to erase

WeddingGuest youngins;

assert(youngins.inviteGuest("Lauren", "U", 23));//tests invite guest

assert(youngins.inviteGuest("James", "H", 29));

assert(youngins.guestCount() == 2);//tests guest count

string first, last;

int a;

assert(youngins.verifyGuestOnTheList(0, first, last, a) && a//tests verifyGuestsOnTheList of first index

== 29);

assert(youngins.verifyGuestOnTheList(1, first, last, a) &&//tests verifyGuestsOnList of last index

(first == "Lauren" && a == 23));

WeddingGuest groomsmen;

groomsmen.inviteGuest("Tony", "Ambrosio", 40);

groomsmen.inviteGuest("Long", "Le", 41);

groomsmen.inviteGuest("Mike", "Wu", 43);

groomsmen.inviteGuest("Robert", "Wells", 44);

groomsmen.inviteGuest("Justin", "Sandobal", 37);

groomsmen.inviteGuest("Nelson", "Villaluz", 38);

WeddingGuest bridesmaids;

bridesmaids.inviteGuest("Serra", "Park", 39);

bridesmaids.inviteGuest("Saadia", "Parker", 37);

assert(!bridesmaids.invitedToTheWedding("", ""));//tests invitedToWedding when guest isnt invited

bridesmaids.inviteGuest("Patricia", "Kim", 39);

bridesmaids.inviteGuest("", "", 21);

bridesmaids.inviteGuest("Kristin", "Livingston", 38);

assert(bridesmaids.invitedToTheWedding("", ""));//tests invitedtoWedding when guest is invited

bridesmaids.crossGuestOff("Patricia", "Kim");

assert(bridesmaids.guestCount() == 4

&& bridesmaids.invitedToTheWedding("Serra", "Park")//tests invitedToWedding

&& bridesmaids.invitedToTheWedding("Saadia", "Parker")

&& bridesmaids.invitedToTheWedding("Kristin", "Livingston")

&& bridesmaids.invitedToTheWedding("", ""));

//checks alphabetical order

//groomsmen.dump();

//bridesmaids.dump();

//test crossGuestOff

groomsmen.crossGuestOff("Robert", "Wells");

groomsmen.crossGuestOff("Mike", "Wu");

assert(!groomsmen.crossGuestOff("Cynthia", "Nguyen"));

assert(groomsmen.guestCount() == 4);

//tests cross all guests off

WeddingGuest test0;

GuestType value;

test0.inviteGuest("test", "test", 0);

test0.inviteGuest("test1", "test1", 0);

test0.inviteGuest("test2", "test2", 0);

while (!test0.noGuests())//tests noGuests

{

test0.verifyGuestOnTheList(0, first, last, value);

test0.crossGuestOff(first, last);

}

assert(test0.noGuests());//tests noGuests

//cerr << "pass test 0" << endl;

//tests swapWeddingGuests

groomsmen.swapWeddingGuests(bridesmaids);

assert(groomsmen.guestCount() == 4

&& groomsmen.invitedToTheWedding("Serra", "Park")

&& groomsmen.invitedToTheWedding("Saadia", "Parker")

&& groomsmen.invitedToTheWedding("Kristin", "Livingston")

&& groomsmen.invitedToTheWedding("", ""));

//groomsmen.dump();

//bridesmaids.dump();

bridesmaids.swapWeddingGuests(groomsmen);

assert(bridesmaids.guestCount() == 4

&& bridesmaids.invitedToTheWedding("Serra", "Park")

&& bridesmaids.invitedToTheWedding("Saadia", "Parker")

&& bridesmaids.invitedToTheWedding("Kristin", "Livingston")

&& bridesmaids.invitedToTheWedding("", ""));

//groomsmen.dump();

//bridesmaids.dump();

//cerr << "pass swap test" << endl;

//tests copy constructor, assignment operator

string first1;

string last1;

GuestType value1;

WeddingGuest temp = groomsmen;//cpy constructor

assert(groomsmen.verifyGuestOnTheList(0, first, last, value) == temp.verifyGuestOnTheList(0, first1, last1, value1) && groomsmen.guestCount() == temp.guestCount());

groomsmen = bridesmaids;//assignment operator

assert(groomsmen.verifyGuestOnTheList(0, first, last, value) == bridesmaids.verifyGuestOnTheList(0, first1, last1, value1) && groomsmen.guestCount() == bridesmaids.guestCount());

groomsmen = temp;

//cerr << "pass constructor and assignment operator" << endl;

bridesmaids.inviteGuest("Tony", "Ambrosio", 0);

//tests matchInvitedGuest && alterGuest

assert(bridesmaids.alterGuest("Tony", "Ambrosio", 43) && bridesmaids.matchInvitedGuest("Tony", "Ambrosio", value) && value == 43);

assert(!bridesmaids.alterGuest("Cynthia", "Nguyen", 20));

//test inviteOrAlter

assert(bridesmaids.inviteOrAlter("Cynthia", "Nguyen", 19) && bridesmaids.guestCount() == 6);

assert(bridesmaids.inviteOrAlter("Tony", "Ambrosio", 60) && bridesmaids.matchInvitedGuest("Tony", "Ambrosio", value) && value == 60 && bridesmaids.guestCount() == 6);

//tests nonmember functions//

//tests joinGuests

WeddingGuest test1;

joinGuests(groomsmen, bridesmaids, test1);//odJined is empty

joinGuests(groomsmen, bridesmaids, temp);//odJoined containes values

assert(test1.verifyGuestOnTheList(0, first, last, value) == temp.verifyGuestOnTheList(0, first1, last1, value1) && test1.guestCount() == temp.guestCount());

attestGuests("\*", "\*", test1, temp);//both first and last are wild

//temp.dump();

WeddingGuest test2;

WeddingGuest test2\_1;

test2.inviteGuest("Anthony", "Nguyen", 18);

test2.inviteGuest("Cynthia", "Nguyen", 19);

test2.inviteGuest("Phuong", "Tran", 76);

test2.inviteGuest("Khanh", "Nguyen", 64);

test2.inviteGuest("Kelvin", "Thoi", 23);

test2.inviteGuest("Cynthia", "Tran", 19);

attestGuests("\*", "Nguyen", test2, temp);//test only first is wild and odResult is not empty

attestGuests("\*", "Nguyen", test2, test2\_1);//test only first is wild and odResult is empty

//temp.dump()

//test2\_1.dump()

assert(temp.guestCount() == test2\_1.guestCount());

assert(test2\_1.guestCount() == 3);

attestGuests("Cynthia", "\*", test2, temp);//test only last is wild

assert(temp.guestCount() == 2);

attestGuests("Cynthia", "\*", test2, test2);//tests if both odOne and odResults are same linked list

assert(test2.guestCount() == 2 && test2.invitedToTheWedding("Cynthia", "Nguyen") && test2.invitedToTheWedding("Cynthia", "Tran"));

return;

}

void teststring() {

// default constructor

WeddingGuest lal;

// For an empty list:

assert(lal.guestCount() == 0); // test size

assert(lal.noGuests()); // test empty

assert(!lal.crossGuestOff("Malik", "Monk")); // nothing to erase

//all tets run with GuestType == string

WeddingGuest youngins;

assert(youngins.inviteGuest("Lauren", "U", "23"));//tests invite guest

assert(youngins.inviteGuest("James", "H", "29"));

assert(youngins.guestCount() == 2);//tests guest count

string first, last;

string a;

assert(youngins.verifyGuestOnTheList(0, first, last, a) && a//tests verifyGuestsOnTheList of first index

== "29");

assert(youngins.verifyGuestOnTheList(1, first, last, a) &&//tests verifyGuestsOnList of last index

(first == "Lauren" && a == "23"));

WeddingGuest groomsmen;

groomsmen.inviteGuest("Tony", "Ambrosio", "40");

groomsmen.inviteGuest("Long", "Le", "41");

groomsmen.inviteGuest("Mike", "Wu", "43");

groomsmen.inviteGuest("Robert", "Wells", "44");

groomsmen.inviteGuest("Justin", "Sandobal", "37");

groomsmen.inviteGuest("Nelson", "Villaluz", "38");

WeddingGuest bridesmaids;

bridesmaids.inviteGuest("Serra", "Park", "39");

bridesmaids.inviteGuest("Saadia", "Parker", "37");

assert(!bridesmaids.invitedToTheWedding("", ""));//tests invitedToWedding when guest isnt invited

bridesmaids.inviteGuest("Patricia", "Kim", "39");

bridesmaids.inviteGuest("", "", "21");

bridesmaids.inviteGuest("Kristin", "Livingston", "38");

assert(bridesmaids.invitedToTheWedding("", ""));//tests invitedtoWedding when guest is invited

bridesmaids.crossGuestOff("Patricia", "Kim");

assert(bridesmaids.guestCount() == 4

&& bridesmaids.invitedToTheWedding("Serra", "Park")//tests invitedToWedding, all invited is in list

&& bridesmaids.invitedToTheWedding("Saadia", "Parker")

&& bridesmaids.invitedToTheWedding("Kristin", "Livingston")

&& bridesmaids.invitedToTheWedding("", ""));

//checks alphabetical order

//groomsmen.dump();

//bridesmaids.dump();

//test crossGuestOff

groomsmen.crossGuestOff("Robert", "Wells");

groomsmen.crossGuestOff("Mike", "Wu");

assert(!groomsmen.crossGuestOff("Cynthia", "Nguyen"));//no guest to cross off

assert(groomsmen.guestCount() == 4);

//tests cross all guests off

WeddingGuest test0;

GuestType value;

test0.inviteGuest("test", "test", "0");

test0.inviteGuest("test1", "test1", "0");

test0.inviteGuest("test2", "test2", "0");

while (!test0.noGuests())//tests noGuests

{

test0.verifyGuestOnTheList(0, first, last, value);

test0.crossGuestOff(first, last);

}

assert(test0.noGuests());//tests noGuests

//cerr << "pass test 0" << endl;

//tests swapWeddingGuests

groomsmen.swapWeddingGuests(bridesmaids);

assert(groomsmen.guestCount() == 4

&& groomsmen.invitedToTheWedding("Serra", "Park")

&& groomsmen.invitedToTheWedding("Saadia", "Parker")

&& groomsmen.invitedToTheWedding("Kristin", "Livingston")

&& groomsmen.invitedToTheWedding("", ""));

//groomsmen.dump();

//bridesmaids.dump();

bridesmaids.swapWeddingGuests(groomsmen);

assert(bridesmaids.guestCount() == 4

&& bridesmaids.invitedToTheWedding("Serra", "Park")

&& bridesmaids.invitedToTheWedding("Saadia", "Parker")

&& bridesmaids.invitedToTheWedding("Kristin", "Livingston")

&& bridesmaids.invitedToTheWedding("", ""));

//groomsmen.dump();

//bridesmaids.dump();

//cerr << "pass swap test" << endl;

//tests copy constructor, assignment operator

string first1;

string last1;

GuestType value1;

WeddingGuest temp = groomsmen;//cpy constructor

assert(groomsmen.verifyGuestOnTheList(0, first, last, value) == temp.verifyGuestOnTheList(0, first1, last1, value1) && groomsmen.guestCount() == temp.guestCount());

groomsmen = bridesmaids;//assignment operator

assert(groomsmen.verifyGuestOnTheList(0, first, last, value) == bridesmaids.verifyGuestOnTheList(0, first1, last1, value1) && groomsmen.guestCount() == bridesmaids.guestCount());

groomsmen = temp;

//cerr << "pass constructor and assignment operator" << endl;

bridesmaids.inviteGuest("Tony", "Ambrosio", "0");

//tests matchInvitedGuest && alterGuest

assert(bridesmaids.alterGuest("Tony", "Ambrosio", "43") && bridesmaids.matchInvitedGuest("Tony", "Ambrosio", value) && value == "43");

assert(!bridesmaids.alterGuest("Cynthia", "Nguyen", "20"));//no guest to alter

//test inviteOrAlter

assert(bridesmaids.inviteOrAlter("Cynthia", "Nguyen", "19") && bridesmaids.guestCount() == 6); //not on list is invited

assert(bridesmaids.inviteOrAlter("Tony", "Ambrosio", "60") && bridesmaids.matchInvitedGuest("Tony", "Ambrosio", value) && value == "60" && bridesmaids.guestCount() == 6); //is on list is altered

//tests nonmember functions//

//tests joinGuests

WeddingGuest test1;

joinGuests(groomsmen, bridesmaids, test1);//odJined is empty

joinGuests(groomsmen, bridesmaids, temp);//odJoined containes values

assert(test1.verifyGuestOnTheList(0, first, last, value) == temp.verifyGuestOnTheList(0, first1, last1, value1) && test1.guestCount() == temp.guestCount());

attestGuests("\*", "\*", test1, temp);//both first and last are wild

//temp.dump();

WeddingGuest test2;

WeddingGuest test2\_1;

test2.inviteGuest("Anthony", "Nguyen", "18");

test2.inviteGuest("Cynthia", "Nguyen", "19");

test2.inviteGuest("Phuong", "Tran", "76");

test2.inviteGuest("Khanh", "Nguyen", "64");

test2.inviteGuest("Kelvin", "Thoi", "23");

test2.inviteGuest("Cynthia", "Tran", "19");

attestGuests("\*", "Nguyen", test2, temp);//test only first is wild and odResult is not empty

attestGuests("\*", "Nguyen", test2, test2\_1);//test only first is wild and odResult is empty

//temp.dump()

//test2\_1.dump()

assert(temp.guestCount() == test2\_1.guestCount());

assert(test2\_1.guestCount() == 3);

attestGuests("Cynthia", "\*", test2, temp);//test only last is wild

assert(temp.guestCount() == 2);

attestGuests("Cynthia", "\*", test2, test2);//tests if both odOne and odResults are same linked list

assert(test2.guestCount() == 2 && test2.invitedToTheWedding("Cynthia", "Nguyen") && test2.invitedToTheWedding("Cynthia", "Tran"));

return;

}