Daren Sathasivam

CS30

Professor Ambrosio

Due: 10/07/2022

Project #1 Report

Design:

For my solution to the WeddlingList program, I chose a doubly linked list. I chose a doubly linked list because of my uncertainty in hash tables, but I hope I can return to this project and create a solution utilizing other data structures such as a BST or hash table. With the doubly linked list, each Guest node stores information such as their full name and a specific value. In addition, there are pointers that allow a traversal forward and backwards through the doubly linked list. The ability to traverse through a list easily is one of the reasons why I chose this over data structure. Another reason why I chose to utilize this data structure is the ability to remove or add an element into the list. The ability to sort the list alphabetically and have it sorted as elements are added or removed is another reason why I preferred this over other methods. The list is not circular and the head and tail nodes will point towards a nullptr. In addition, there are no dummy nodes at either end but a node is created at either end if an element is added at the head or tail element.

Obstacles:

I came across a few obstacles throughout the project and the recurring theme around them would be my ability to envision the nodes at the ends of the list and the different cases. For the envisioning of the ends of the list, Corey had helped me during the lab through visuals and helped me understand how the end of the nodes would act during certain cases. The other main obstacle I encountered was figuring out the different cases for the insertion in addition to adding elements in alphabetical order. There were many different possible cases and sorting out the cases through writing pseudocode helped me overcome this obstacle. In addition, most of the other functions require similar cases. So, after figuring out the inviteGuest function, most of the other functions were easier to implement in terms of cases. I am somewhat confident that every case will be met by the proper algorithm to insert elements in a sorted manner.

Pseudocode:

Copy Constructor:

```
-Initialize head and tail ptrs and the two lists
-traverse through other list
    -if the copied list is at nullptr
        -copy info then traverse down other list
    -else
        -copy info into next of copiedl list
        -traverse down other list
-set tail to copied list
```

Assignment Operator:

```
-assigns copy to the right-hand list
-then swap heads of the two lists
```

inviteGuest:

```
-initialize list
```

```
-if last name is less than last name in list or if last names are same but first name is less than first name in list
    -insert guest into doubly linked list utilizing a temp node to help shift elements around
    -return true bc inserting into list
-loop through original list
    -if full name is already on list
        -return false bc already in list
    -if last name is the same
        -if first name is less than first name in list
            -insert and return true
        -continue down list
        -if last name is less than last name in list
            -insert and return true
-insertion for case where tail pointer points to nullptr
    -use temp values to shift nodes around so there is an extra slot for insertion
```

alterGuest:

```
-traverse through list
    -if full name is in list
        -set value to parameter value and return true
```

inviteOrAlter:

-return using inviteGuest or alterGuest function because the two will return boolean values if the case is met for either insertion or altering the value

crossGuestOff:

```
-traverse through list
    -if full name is already on the list
        -set nodes from either end of element to each other so original can be removed from in between the two elements
        -remove element and return true
    -no removal returns false
```

invitedToWedding, matchInvitedGuest, & swapWeddingGuests:

```
-simple/trivial functions that implement similar algorithms from previous functions
```

verifyGuestOnTheList:

```
-traverse through list
    -if an element is within the number of total elements within a list
        -copy values from position i in the list ot the values of the parameters and return true
    -increment to go further down the list
-return false and leave parameters unchanged if not on list
```

joinGuests:

```
-initialize variables
-loop through odOne using verifyGuestOnTheList function
    -check the odJoined list
        -if odOne value is the same as odTwo value, continue
        -return result as false
        -crossGuestOff function for odJoined then continue
    -inviteGuest into odJoined list from odOne elements
```

-reset increment and repeat for odTwo

-return the either true or false depending on whether or not the lists were combined

<u>attestGuests:</u>

-initialize variables

-traverse through odOne using verifyGuestOnTheList function and get the info from given list

    -increment

    -if the values match with the searches, add the guest to the odResult list

Test Cases:

Tests performed on a map from strings to integers
```
// default constructor
WeddingGuest lal;

// For an empty list:
assert(lal.guestCount() == 0);

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

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

Used to check output for most tests without odOne/odTwo/odResult/odJoined:
```
for (int n = 0; n < groomsmen.guestCount(); n++)
{
    string first;
    string last;
    int val;
    groomsmen.verifyGuestOnTheList(n, first, last, val);
    cout << first << " " << last << " " << val << endl;
}

// Copy constructor test
WeddingGuest groomsmen;
groomsmen.inviteGuest("Tony", "Ambrosio", 40);
groomsmen.inviteGuest("Mike", "Wu", 43);
groomsmen.inviteGuest("Robert", "Wells", 44);
groomsmen.inviteGuest("Justin", "Sandobal", 37);
groomsmen.inviteGuest("Nelson", "Villaluz", 38);
groomsmen.inviteGuest("Long", "Le", 41);
```

```
WeddingGuest copy(groomsmen); // call to copy constructor

// Assignment Operator test
WeddingGuest groomsmen;
groomsmen.inviteGuest("Tony", "Ambrosio", 40);
groomsmen.inviteGuest("Mike", "Wu", 43);
groomsmen.inviteGuest("Robert", "Wells", 44);
groomsmen.inviteGuest("Justin", "Sandobal", 37);
groomsmen.inviteGuest("Nelson", "Villaluz", 38);
groomsmen.inviteGuest("Long", "Le", 41);
WeddingGuest copy;
copy = groomsmen; // call to assignment operator

// Destructor test
WeddingGuest groomsmen;
groomsmen.inviteGuest("Tony", "Ambrosio", 40);
groomsmen.inviteGuest("Mike", "Wu", 43);
groomsmen.inviteGuest("Robert", "Wells", 44);
groomsmen.inviteGuest("Justin", "Sandobal", 37);
groomsmen.inviteGuest("Nelson", "Villaluz", 38);
groomsmen.inviteGuest("Long", "Le", 41);
groomsmen.~WeddingList();
assert(lal.noGuests());

// Invite Guest and Verify Guest On The List test
void test() {
    WeddingGuest eliteSingles;
    assert(eliteSingles.inviteGuest("Jackie", "S",
        "jackies@elitesingles.com"));
    assert(eliteSingles.inviteGuest("Mark", "P",
        "markp@elitesingles.com"));
    assert(eliteSingles.guestCount() == 2);
    string first, last, e;
    assert(eliteSingles.verifyGuestOnTheList(0, first, last, e)
        && e == "markp@elitesingles.com");
    assert(eliteSingles.verifyGuestOnTheList(1, first, last, e)
        && (first == "Jackie" && e ==
"jackies@elitesingles.com"));
    return;
}

// Invite Guest with GuestType as double and invitedToTheWedding
test
void test() {
    WeddingGuest bridesmaids;
    bridesmaids.inviteGuest("Serra", "Park", 39.5);
    bridesmaids.inviteGuest("Saadia", "Parker", 37.5);
    assert(!bridesmaids.invitedToTheWedding("", ""));
    bridesmaids.inviteGuest("Patricia", "Kim", 39.0);
    bridesmaids.inviteGuest("", "", 21.0);
    bridesmaids.inviteGuest("Kristin", "Livingston", 38.0);
    assert(bridesmaids.invitedToTheWedding("", ""));
    bridesmaids.crossGuestOff("Patricia", "Kim");
    assert(bridesmaids.guestCount() == 4
        && bridesmaids.invitedToTheWedding("Serra", "Park")
        && bridesmaids.invitedToTheWedding("Saadia", "Parker")
        && bridesmaids.invitedToTheWedding("Kristin",
"Livingston")
        && bridesmaids.invitedToTheWedding("", ""));
}

// Invite Guest with GuestType as int test
WeddingGuest groomsmen;
groomsmen.inviteGuest("Tony", "Ambrosio", 40);
groomsmen.inviteGuest("Mike", "Wu", 43);
groomsmen.inviteGuest("Robert", "Wells", 44);
groomsmen.inviteGuest("Justin", "Sandobal", 37);
groomsmen.inviteGuest("Nelson", "Villaluz", 38);
groomsmen.inviteGuest("Long", "Le", 41);
for (int n = 0; n < groomsmen.guestCount(); n++)
{
    string first;
    string last;
    int val;
    groomsmen.verifyGuestOnTheList(n, first, last, val);
    cout << first << " " << last << " " << val << endl;
}

// Alter Guest and matchInvitedGuest test
WeddingGuest groomsmen;
groomsmen.inviteGuest("Tony", "Ambrosio", 40);
groomsmen.inviteGuest("Mike", "Wu", 43);
groomsmen.inviteGuest("Robert", "Wells", 44);
groomsmen.inviteGuest("Justin", "Sandobal", 37);
groomsmen.inviteGuest("Nelson", "Villaluz", 38);
groomsmen.inviteGuest("Long", "Le", 41);
groomsmen.alterGuest("Long", "Le", 69);
int val;
assert(groomsmen.matchInvitedGuest("Long","Le", val) && val ==
69);
cout << "Test passed.\n";

// Attest Guest with last name as escape sequence test
WeddingGuest odOne, odTwo;
```

```
odOne.inviteGuest("Cobey", "C", 35);
odOne.inviteGuest("Dan", "H", 38);
odOne.inviteGuest("Dan", "V", 44);
odOne.inviteGuest("Dion", "V", 45);
attestGuests("Dan", "*", odOne, odOne);
int count = odOne.guestCount();
for (int i = 0; i < count; i++)
{
    string firstName, lastName;
    GuestType value;
    odOne.verifyGuestOnTheList(i, firstName, lastName, value);
    cout << firstName << " " << lastName << " " << value <<
"\n";
}

// Attest Guest with first name as escape sequence test
WeddingGuest odOne, odTwo;
odOne.inviteGuest("Cobey", "C", 35);
odOne.inviteGuest("Dan", "H", 38);
odOne.inviteGuest("Dan", "V", 44);
odOne.inviteGuest("Dion", "V", 45);
attestGuests("*", "V", odOne, odOne);
int count = odOne.guestCount();
for (int i = 0; i < count; i++)
{
    string firstName, lastName;
    GuestType value;
    odOne.verifyGuestOnTheList(i, firstName, lastName, value);
    cout << firstName << " " << lastName << " " << value <<
"\n";
}

// Attest Guest with first name and last name both as escape
sequence test
WeddingGuest odOne, odTwo;
odOne.inviteGuest("Cobey", "C", 35);
odOne.inviteGuest("Dan", "H", 38);
odOne.inviteGuest("Dan", "V", 44);
odOne.inviteGuest("Dion", "V", 45);
attestGuests("*", "*", odOne, odOne);
int count = odOne.guestCount();
for (int i = 0; i < count; i++)
{
    string firstName, lastName;
    GuestType value;
    odOne.verifyGuestOnTheList(i, firstName, lastName, value);
    cout << firstName << " " << lastName << " " << value <<
"\n";
}

// Join guests with same guests having different values in both
lists test
WeddingGuest odOne, odTwo;
odOne.inviteGuest("Pete", "Best", 3);
odOne.inviteGuest("John", "Lennon", 1);
odOne.inviteGuest("Paul", "McCartney", 2);
odTwo.inviteGuest("Pete", "Best", 6);
odTwo.inviteGuest("George", "Harrison", 4);
odTwo.inviteGuest("Ringo", "Starr", 5);
if (!joinGuests(odOne, odTwo, odOne))
{
    int count = odOne.guestCount();
    for (int i = 0; i < count; i++)
    {
        string firstName, lastName;
        GuestType value;
        odOne.verifyGuestOnTheList(i, firstName, lastName,
value);
        cout << firstName << " " << lastName << " " << value <<
"\n";
    }
}
else
{
    cout << "Failed\n";
}

// Join Guests with same guests and same values in different
lists test
WeddingGuest odOne, odTwo;
odOne.inviteGuest("Anthony", "Davis", 3);
odOne.inviteGuest("Lebron", "James", 23);
odOne.inviteGuest("Malik", "Monk", 11);
odTwo.inviteGuest("Lebron", "James", 23);
odTwo.inviteGuest("Russel", "Westbrook", 0);
if (joinGuests(odOne, odTwo, odOne))
{
    int count = odOne.guestCount();
    for (int i = 0; i < count; i++)
    {
        string firstName, lastName;
        GuestType value;
        odOne.verifyGuestOnTheList(i, firstName, lastName,
value);
        cout << firstName << " " << lastName << " " << value <<
"\n";
    }
}
else
{
    cout << "Failed\n";
}
```