Jesse Narkmanee

805190133

CS 32 Project 2 Report

**Description of Design**

For this project, I created a doubly-linked list that had a head and tail pointer with previous and next in the node struct so that the list can be bi-directional. Overall, I tried to reuse as much of the assigned public member function as much as possible as helper functions in my other implementation to be the simplest and time-saving. Also, if I saw a pattern of implementations that I would later reuse in my project as I was planning on how to tackle other assigned functions, I created private member helper functions to again things help me with efficiency as the specs told me I could do so. Lastly, much of the function required full names, so I programmed it where each node would automatically assign a fullName variable by string appendage after construction to further help make things easier and clear in the code when dealing with just full names.

**Obstacles and Difficulties**

My greatest difficulty came with the assignment operator, copy constructor, and the changeGuestList() function because I add the wrong initial approach. For these exact same functions, I also had trouble implementing in our most previous homework. At first, I treated the link list more like the dynamically allocated rule of three array example in class: assigning private variables, the constructing each element. Then looking back at the homework, I realized that linked lists are special in that we do not need to necessarily copy each and single element but swap the address of the head and tail pointers, making the solution so much easier. After implementing the harder way and getting errors, I did it this way for the assignment operator, and it worked perfectly, teaching me a great deal of the efficiency of pointers sometimes.

**Pseudocode**

// If person is on the guest list, we now need to begin search of the list

// When found we delete the node then reattach other nodes with our helper function

**dropFromGuestList(const std::string& firstName, const std::string& lastName);**

// If the person's full name is already on the guest list, return false.

// Add to front if list is empty or if alphabetically before values of first node

// In all other cases we need to find the above our targeted insertion place

// After being in position, we create our node and properly link our previous and next pointers of both of the node to be inserted and other disconnected nodes

**addInvitee(const std::string& firstName, const std::string& lastName, const BirthdayType& value);**

// Iterate through the second list with the range of number of its guest list

// Each iteration (i) selects a variable to pass back by reference into addInvitee()

// If this node full name is also in the other object, return false later

// If not yet on the list, then add it to bpJoined, if already on the list then do nothing

// Iterate through the second list with the range of number of its guest list

// Each iteration (i) selects a variable to pass back by reference into addInvitee()

// If this node full name is also in the other object, return false later

// If not yet on the list, then add it to bpJoined, if already on the list then do nothing

**combineGuestLists(const BirthdayParty & bpOne, const BirthdayParty & bpTwo, BirthdayParty & bpJoined);**

// Assignment Operator is called if both wildcarded to make a copy

// Iterate through the second list with the range of number of its guest list

// Each iteration (i) selects a variable to pass back by reference into addInvitee()

**verifyGuestList (const std::string& fsearch, const std::string& lsearch, const BirthdayParty& bpOne, BirthdayParty& bpResult);**

// To protect self-assignment, anything equal to itself is itself.

// A temporary object placeholder on the stack so others do not need to change

// Calling a helper function which swaps linked lists

// We return \*this to ensure chaining assignments will work by returning a reference to itself

operator=(const BirthdayParty& other);

**Test Cases**

  // Different Input Variables for Linked List Objects

  /\*int age = 21;

  string firstName = "Jesse";

  string lastName = "Narkmanee";

  string firstName2 = "James";

  string lastName2 = "Moseley";

  int age2 = 30;

  string firstName3 = "Jonathan";

  string lastName3 = "Chu";

  int age3 = 21;

  string firstName4 = "Yin-Yeung";

  string lastName4 = "Cheung";

  int age4 = 21;

  string firstName5 = "Yanni";

  string lastName5 = "Backster";

  int age5 = 25;

  string firstName6 = "Zachary";

  string lastName6 = "Bentley";

  int age6 = 37;

  cerr << endl << "///////////////////////////////////////////////////////////////////////////////////" << endl;

  // Shows if my full name ordering function works since Moseley is before Narkmanee

  cerr << isFirstAlpha(firstName, lastName, firstName2, lastName2) << endl; // Works

  cerr << endl << "///////////////////////////////////////////////////////////////////////////////////" << endl;

  // Testing Front Appendage of Linked List Algorithm

  // addToFront(firstName, lastName, age); // Works

  // addToFront(firstName2, lastName2, age2); // Works

  // printValues(); // Works

  cerr << endl << "///////////////////////////////////////////////////////////////////////////////////" << endl;

  // Shows if Invitee will be added if already there (it should't) and if it is in order

  addInvitee(firstName, lastName, age); // Works

  addInvitee(firstName3, lastName3, age3); // Works

  addInvitee(firstName4, lastName4, age4); // Works

  addInvitee(firstName2, lastName2, age2); // Works

  printValues(); // Works

  cerr << endl << "///////////////////////////////////////////////////////////////////////////////////" << endl;

  // Shows number of Invitees and if how many are on the guest list is correct

  cerr << noInvitees() << endl; // Works

  cerr << whosOnTheGuestList() << endl; // Works

  cerr << endl << "///////////////////////////////////////////////////////////////////////////////////" << endl;

  // Shows if I can modify the value of Jesse Narkmanee

  cerr << modifyInvitee(firstName, lastName, age2) << endl; // Works

  printValues();

  cerr << endl << "///////////////////////////////////////////////////////////////////////////////////" << endl;

  // Shows if I can sucessfully delete Jesse Narkmanee from list

  cerr << dropFromGuestList(firstName, lastName) << endl; // works

  printValues();

  cerr << endl << "///////////////////////////////////////////////////////////////////////////////////" << endl;

  // Shows if I am able to add or modify under certain conditions

  cerr << "addOrModify();" << endl;

  cerr << addOrModify(firstName3, lastName3, age2) << endl; // error

  printValues();

  cerr << endl << "///////////////////////////////////////////////////////////////////////////////////" << endl;

  // Shows if I do select the first index of i

  int i = 1;

  cout << selectInvitee(i, firstName, lastName, age) << endl; // Works

  printValues();

  cerr << endl << "///////////////////////////////////////////////////////////////////////////////////" << endl;

  // To prepare for the next tests

  BirthdayParty other;

  other.addToFront(firstName4, lastName4, age4);

  other.addToFront(firstName3, lastName3, age3);

  other.addToFront(firstName5, lastName5, age5);

  other.addToFront(firstName6, lastName6, age6);

  other.printValues();

  cerr << endl << "///////////////////////////////////////////////////////////////////////////////////" << endl;

  // See if I am able to swap guest lists

  changeGuestList(other);

  printValues();

  cout << endl << "Other:" << '\n';

  other.printValues();

  cerr << endl << "///////////////////////////////////////////////////////////////////////////////////" << endl;

  // Continuation of Previous

  BirthdayType value;

  cerr << checkGuestList(firstName3, lastName3, value) << endl;

  cout << endl << "Original:" << '\n';

  printValues();

  cerr << "Passed by reference is now: " << value << endl;

  cerr << endl << "///////////////////////////////////////////////////////////////////////////////////" << endl;

  // Continuation of Previous

  BirthdayParty bpOne;

  bpOne.addToFront(firstName, lastName, age); // Works

  bpOne.addToFront(firstName2, lastName2, age2); // Works

  bpOne.addToFront(firstName3, lastName3, age3); // Works

  bpOne.addToFront(firstName4, lastName4, age4); // Works

  other.addToFront(firstName3, lastName3, age3);

  other.addToFront(firstName5, lastName5, age5);

  other.addToFront(firstName6, lastName6, age6);

  cout << endl << "bpOne:" << '\n';

  bpOne.printValues();

  cout << endl << "Other:" << '\n';

  other.printValues();

  BirthdayParty bpJoined;

  combineGuestLists(bpOne, other, bpJoined);

  cout << endl << "bpJoined:" << '\n';

  bpJoined.printValues();

  cerr << endl << "///////////////////////////////////////////////////////////////////////////////////" << endl;

  // Checks to see if I can search if we have one or both wildcards (should be just a copy in latter case)

  BirthdayParty bpResult;

  verifyGuestList ("\*", "\*", bpOne, bpResult);

  cout << endl << "bpResult:" << '\n';

  bpResult.printValues();

  cerr << endl << "///////////////////////////////////////////////////////////////////////////////////" << endl;\*/