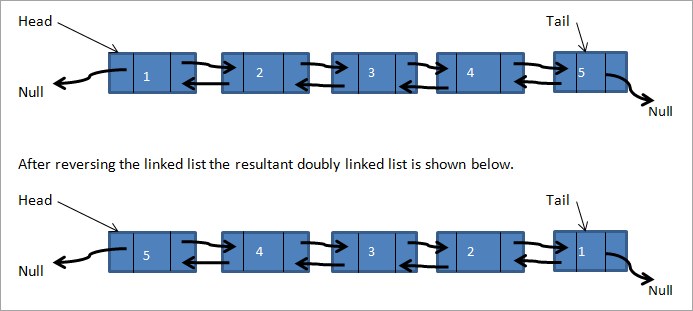
**Implementation Description**

I just used a simple doubly linked list for my implementation. No dummy nodes were used, and it is not circularly linked. I incorporated a tail pointer in addition to the head pointer in order to keep track of the last node, and to have the ability to loop through the nodes in reverse order. Each node in the list contains a first name, a last name, a BirthdayType item, a next pointer, and a previous pointer.



In the diagram above, the sections that are numbered contain the first name, the last name, and the BirthdayType item of that node.

**Obstacles**

Writing a program this large was difficult for me, because I had to keep track of a lot of different factors. I especially spent a lot of time on making sure that there were no memory leaks. The addition of the previous pointer was another obstacle because I would link nodes through the next pointer and then forget to link the previous pointers. Also, keeping special cases in mind, such as the list being empty or having only one element, was difficult while writing all of the functions. I had problems in the non-member functions because I wasn’t used to writing code that wasn’t able to access the private members of the class, but I figured out how to use the public member functions appropriately.

**Pseudocode**

*BirthdayParty Constructor (create empty list):*

Set head and tail pointers to null because there are no nodes

*BirthdayParty Copy Constructor*

If rhs is empty

Initialize head and tail

Else

Set values for the first node

Create temporary pointers for traversal

Loop through rhs nodes

Allocate memory for new node

Copy values into node

Set next and prev pointers

Increment to next nodes

*BirthdayParty Destructor:*

Create temp pointer to loop through nodes

Create temp pointer to hold location of next pointer after deletion

Loop through nodes w/ first temp pointer

Set second temp pointer to next node

Delete current node

Set pointer for deleted node to temp

*BirthdayParty Assignment Operator (adapted from Prof. Ambrosio’s hw1 solution):*

Check if lists are already equal

Make copy of rhs and switch lists

Return \*this

*BirthdayParty addInvitees*

Create temp pointers to help loop through (p and temp)

Create bool before to signify where node should be inserted

Create bool found to help with ending loops

If list is empty

Allocate memory

Set values and link

Set head and tail

Return true

Loop through nodes using pointer p

If the current node’s last name is greater than lastName

Set before to true (node should be placed before this node)

Set found to true

Loop through nodes where the last names are equal and node hasn’t been found

If the first names are the same

Return false

If the first name is greater than firstName

Set before to true (node should be placed before this node)

Set found to true

Else if the next node is null

Set before to false (node should be placed after this node)

Set found to true

If found is false

Increment to next node

If found is true or the next node is null

Break

Increment p

Allocate new node and set temp pointer to it

Copy values and set node next and prev pointers to null

If before is false

Link the temp node’s previous pointer to p

If there is a valid node after temp

Set temp’s next

Set previous pointer of the node after temp

Otherwise it is the last node so

Set tail to temp

Link the previous pointer’s (p) next to temp

Else if before is true

Link temp’s node next pointer to p

If there is a valid node before temp

Set temp’s prev

Set next pointer of the node before temp

Otherwise it is the first node so

Set head to temp

Link the following pointer’s (p) prev to temp

Return true

*BirthdayParty dropFromGuestList*

Check if list is empty

If it matches the first node

Temporary target pointer to hold location

If there is only one node in list

Set head and tail to null

Else

Set head to node after target

Change next node prev pointer to null

Delete target pointer

Return true

Create temporary pointer

Loop through

If next node matches full name

Break (stop temp pointer at node before target)

Increment

If pointer is not null, we found position

Create target pointer and set to node after temp pointer

Link previous node to following node

If next node exists

Link following node to previous node

Else (it is the last node)

Set tail to temp pointer

Delete target pointer

Return true

Return false

*combineGuestLists*

Create temporary list and set equal to resulting list

Find current size of result

If size is not zero

Loop through nodes

Delete each node

Find size of first list

Find size of second list

Create counter integer

Create bool add to verify item should be added to list

Create bool return1 to determine return value

Loop through second lists nodes

Set counter to 0

Create empty values for first, last, and value for both lists

Use selectInvitee to get information for second list

If first list contains the first and last name of the second list’s current node

Loop through first lists nodes using counter and size

Use selectInvitee to get values for first list

If the full names are the same but the values are different

Set add to false

Set return1 to false

Increment counter

If add is true

Add invitee to temp list

Reset add to true

Loop through first lists nodes

Set counter to 0

Create empty values for first, last, and value for both lists

Use selectInvitee to get information for first list

If second list contains the first and last name of the first list’s current node

Loop through second lists nodes using counter and size

Use selectInvitee to get values for second list

If the full names are the same but the values are different

Set add to false

Set return1 to false

Increment counter

If person isn’t already in temp list and add is true

Add invitee to temp list

Reset add to true

Set resulting list equal to temp list

If bool return1 is false

Return false

Return true

*verifyGuestList*

Create temporary list and set equal to resulting list

Find current size of result

If size is not zero

Loop through nodes

Delete each node

Loop through nodes of given list

Create empty values for first, last, and value for both lists

Use selectInvitee to get information for second list

If full name matches search

Add invitee

Else if first name matches search and last name is \*

Add invitee

Else if first name is \* and last name matches search

Add invitee

Else if both search entries are \*

Add invitee

Set resulting list to temp

**Test Cases**

1. ***Test the nontrivial functions alongside addInvitee***

BirthdayParty theLastDance;

theLastDance.addInvitee("Michael", "Jordan", 23);

theLastDance.addInvitee("Scottie", "Pippen", 33);

theLastDance.addInvitee("Dennis", "Rodman", 91);

theLastDance.addInvitee("Luc", "Longley", 13);

theLastDance.addInvitee("Ron", "Harper", 9);

for (int n = 0; n < theLastDance.whosOnTheGuestList(); n++)

{

string first;

string last;

int val;

theLastDance.selectInvitee(n, first, last, val);

cout << first << " " << last << " " << val << endl;

}

This code segment tests if **addInvitee** alphabetizes the invitees correctly. It also checks if **whosOnTheGuestList** returns the correct number because if all of the names are printed, then it looped through all of the nodes. It then checks the **selectInvitee** function by correctly returning the first name, last name, and BirthdayType item.

theLastDance.addInvitee("Michael", "Jordan", 23);

theLastDance.addInvitee("Scottie", "Jordan", 33);

theLastDance.addInvitee("Dennis", "Jordan", 91);

theLastDance.addInvitee("Luc", "Longley", 13);

theLastDance.addInvitee("Ron", "Harper", 9);

I changed the code segment to have duplicate last names to check the that it is alphabetized by first names.

theLastDance.addInvitee("", "", 23);

theLastDance.addInvitee("", "", 33);

theLastDance.addInvitee("Dennis", "Rodman", 91);

theLastDance.addInvitee("Luc", "Longley", 13);

theLastDance.addInvitee("Ron", "Harper", 9);

Then I changed the code segment to have two entries with empty strings to ensure that the first empty name will be printed and that the second one won’t be, ensuring that the check for duplicates works.

if (theLastDance.personOnGuestList("Michael", "Jordan"))

cout << "he exists" << endl;

if (!theLastDance.personOnGuestList("Nikita", "Patra"))

cout << "who?" << endl;

To check **personOnGuestList**, I looked for someone who was in the list and someone who wasn’t.

int x = 3;

int y = 3;

theLastDance.checkGuestList("Ron", "Harper", x);

theLastDance.checkGuestList("Nikita", "Patra", y);

cout << x << endl;

cout << y << endl;

To check **checkGuestList**, I first called the function on a member that did exist on the list and ensured that the value parameter was changed, and then called it on a member that didn’t exist on the list and ensured that the value remained unchanged.

BirthdayParty theLastDance;

if (theLastDance.noInvitees())

cout << "empty" << endl;

else

cout << "check your code idiot" << endl;

To check **noInvitees**, I called the function on the original list to make sure it returned false, and then changed the list to be empty and called the function to see if it returned true.

1. ***Test modifyInvitee, addOrModify***

BirthdayParty theLastDance;

theLastDance.addInvitee("Michael", "Jordan", 23);

theLastDance.addInvitee("Scottie", "Pippen", 33);

theLastDance.addInvitee("Dennis", "Rodman", 91);

theLastDance.addInvitee("Luc", "Longley", 13);

theLastDance.addInvitee("Ron", "Harper", 9);

int x = 3;

if (theLastDance.modifyInvitee("Ron", "Harper", x))

cout << "true" << endl;

if (!theLastDance.modifyInvitee("Nikita", "Patra", x))

cout << "false" << endl;

for (int n = 0; n < theLastDance.whosOnTheGuestList(); n++)

{

string first;

string last;

int val;

theLastDance.selectInvitee(n, first, last, val);

cout << first << " " << last << " " << val << endl;

}

To check **modifyInvitee,** I called it on a member that does exist in the list; I then printed the list to check that the value changed, and I printed whether it returned true. I then did the opposite case, where I called the function on a member that doesn’t exist, and ensured that the function returned false.

int x = 3;

if (theLastDance.addOrModify("Ron", "Harper", x))

cout << "true" << endl;

if (theLastDance.addOrModify("Nikita", "Patra", x))

cout << "true" << endl;

I then changed the middle code segment from above to test for **addOrModify**. I called it on an existing name and noted that the value changed and it returned true; then, I called it on a nonexistent name and noted that it was added alphabetically to the list and it returned true.

1. ***Test dropFromGuestList, changeGuestList***

BirthdayParty theLastDance;

theLastDance.addInvitee("Michael", "Jordan", 23);

theLastDance.addInvitee("Scottie", "Pippen", 33);

theLastDance.addInvitee("Dennis", "Rodman", 91);

theLastDance.addInvitee("Luc", "Longley", 13);

theLastDance.addInvitee("Ron", "Harper", 9);

if (theLastDance.dropFromGuestList("Ron", "Harper"))

cout << "true" << endl;

if (!theLastDance.dropFromGuestList("Nikita", "Patra"))

cout << "false" << endl;

for (int n = 0; n < theLastDance.whosOnTheGuestList(); n++)

{

string first;

string last;

int val;

theLastDance.selectInvitee(n, first, last, val);

cout << first << " " << last << " " << val << endl;

}

I called **dropFromGuestList** on a name that already exists and on that doesn’t. I then checked the return value and printed the list to make sure that the node was removed.

BirthdayParty check1;

check1.addInvitee("Kobe", "Bryant", 8);

check1.addInvitee("AC", "Green", 45);

check1.addInvitee("AC", "Green", 34);

theLastDance.changeGuestList(check1);

cout << "lastDance: " << endl;

for (int n = 0; n < theLastDance.whosOnTheGuestList(); n++)

{

string first;

string last;

int val;

theLastDance.selectInvitee(n, first, last, val);

cout << first << " " << last << " " << val << endl;

}

cout << "check1: " << endl;

for (int n = 0; n < check1.whosOnTheGuestList(); n++)

{

string first;

string last;

int val;

check1.selectInvitee(n, first, last, val);

cout << first << " " << last << " " << val << endl;

}

I created a new list and used **changeGuestList** to switch check 1 with theLastDance from the previous code segments. I then printed both lists to make sure every element was swapped.

1. ***Check combineGuestLists***

BirthdayParty check1;

check1.addInvitee("Kobe", "Bryant", 8);

check1.addInvitee("AC", "Green", 45);

check1.addInvitee("AC", "Green", 34);

BirthdayParty check2;

check2.addInvitee("Kobe", "Bryant", 24);

check2.addInvitee("Idiot", "Dummy", 16);

check2.addInvitee("Horace", "Grant", 54);

check2.addInvitee("Shaq", "Oneal", 34);

BirthdayParty check3;

check3.addInvitee("Horace", "Grant", 60);

if (combineGuestLists(check1, check2, check1))

cout << "true" << endl;

else

cout << "false" << endl;

check1.printList();

I first wrote a printList function for convenience purposes for further test cases for **combineGuestLists**. I then created three new lists; an example of one of the test cases I used is above. I checked for duplicates between check1 and check2 and made sure that it was only printed once in the resulting list. I made sure that check3’s old values were erased. I then changed duplicate names to have different values and made sure it didn’t print in the resulting list and returned false. I repeated the above process but with empty strings as well. I then inputted parameters that were both a list to be combined and the joint list (see code parameters above for example), and made sure the output was correct.

1. ***Test verifyGuestList***

BirthdayParty result = check3;

BirthdayParty a;

a.addInvitee("Kobe", "Bryant", 8);

a.addInvitee("Gianna", "Bryant", 45);

a.addInvitee("Shaq", "Oneal", 34);

verifyGuestList("\*", "Bryant", a, result);

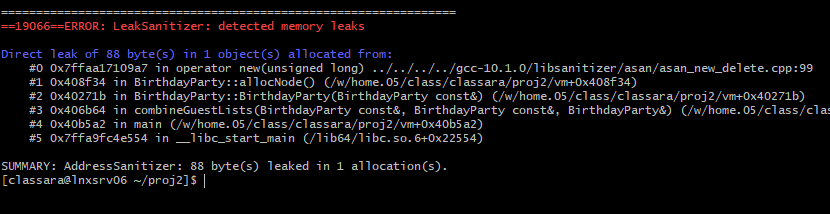
result.printList();

The above code is the structure I used for multiple test cases. At first, I created an empty BirthdayParty and another with nodes, and then called **verifyGuestList**. After the output was correct, I inputted different strings for the search terms to ensure all conditions were met. I then set the list result to check3, a list from previous test cases, and then made sure that it was emptied before adding the relevant nodes.

1. ***Run in g32***

Also add in the other tests from the Project2 spec

1. ***Test for memory leaks in g32***



In the Linux text editor, I created temporary functions allocNode() and freeNode() which would create a node and delete a node, respectively. I created a counter variable and would increment it in allocNode and the decrement in freeNode. Then, I replaced every place I had created a new Node with the allocNode function and every place I had deleted a Node with the freeNode function. I also printed the counter variable in the destructor, and set it to 0 in the constructor and copy constructor. With these adjustments, I could run my code and I could track where I was allocating memory and where I was freeing it, ensuring that in the end, counter should be 0. Just to make sure that the functions I implemented worked correctly I commented out a line that utilized the freeNode function, and the counter no longer printed 0 and the compiler gave me an error message for a memory leak (pictured above), confirming that my functions worked. With this method, I checked for memory leaks in all of the functions that allocated and deleted memory (**constructors, destructors**, etc). combineGuestList and verifyGuestList also relied on the **assignment operator** to work, so checking for memory leaks while testing these functions helped ensure that it worked.