Homework Assignment 3

In this homework, you need to design a linked list to keep track of a telephone directory. Each node in the list holds a person's name and phone number. The nodes in the list are sorted by a person's name. You need to design two classes:

- Class *Node* has the following **private** data:

```
string name;
string phoneNumber;
```

- Class *LL* has the following **private** data:

```
Node * head; // head pointer points to beginning of linked list
```

  Class *LL* should be made friend of class *Node* so public functions of *LL* can access a node's private attributes without having to write getters and setters for class *Node*.

  You also need to implement the following public methods for class *LL*:

  - `LL::LL()`
    The default constructor sets the head to nullptr

  - `void LL::readFromArrays(string nArr[], string pArr [], int size)`
    This function reads two arrays, nArr and pArr, which contain some people's names and corresponding phone numbers, and merges the data of both arrays into a linked list of nodes. Each node holds the name and phone number of one person. Assume the list is initially empty. You should link the nodes in the same order as the elements in the arrays. Which means, the first name is stored in the first node of the list, the second name in the second node and so on.

  - `void LL::insert(string pName, string phone)`
    This function creates a node, initializes its data to the values sent as arguments, and inserts it in a way to maintain the linked list order

  - `void LL::insertAtPos(string pName, string phone, int pos)`
    This function creates a node, initializes its data to the values sent as arguments, and inserts it at position pos in the list. If pos is 1, insert it as the first node. If pos is 2, insert it as the second node, and so on..

  - `void LL::print()`
    This function traverses the list printing the data in each node preceded by the node number. For example:

```
1. Adam James, 202-872-1010
2. Henry Hatch, 617-227-5452
3.
```

…

- void LL::printReverse()
  This function prints the data in each node in reverse order. In other words, it prints the data of the last node first, followed by the data of the second to last node and so on… There is no need to print the record/node number

- void LL::searchByName(string pName)
  This function searches for a particular person's record in the list, whose name is pName. If it finds a record for this person, it prints the **node number** or position of that record in the list. The first node in the list is at position 1, the second node at position 2, and so on. If no matching record exists, the function prints an error message

- void LL::updateNumber(string pName, string newPhone)
  This function updates a specific person's record. If there exists a person whose name is pName, then the corresponding phone number should be updated to newPhone. If no such person is found, an error message is displayed

- void LL::removeRecord(string pName)
  This functions searches for a specific person's record, whose name is pName, and removes it from the list if it exists. Otherwise, an error message is displayed

- LL::~LL()
  The destructor destroys (deletes) all nodes in the list and sets head back to null. Have the destructor call another private function destroy that does the deleting. You need to implement that function too:

  void LL:destroy()

- LL::LL(const LL& source)
  The copy constructor performs a deep copy of the list named source

- LL& LL::operator=(const LL& source)
  This function overloads the assignment operator to perform a deep copy of the list named source. Make sure to free existing nodes in list (calling object) before making its head point to a copy of the list source

- void LL::reverse()
  This function rearranges the nodes in the list so that their order is reversed (stored in descending order of names). It traverses the list and reverses the list nodes so the last node becomes the first and the first node becomes the last one

After designing your classes, you can test your functions using the main program below:

```
const int SIZE = 3;
int main()
{
      string nameArray [SIZE] = {“Jim Meyer”, “Joe Didier”, “Mayssaa
      Najjar”};
      string phoneArray [SIZE] = {“337-465-2345”, “352-654-1983”,
      “878-635-1234”};

      // creating list1 using default constructor
      LL list1;

      // reading from arrays into list 1
      cout << “----------------------------------------” << endl;
      cout << “Reading from arrays into list and printing list1”
      << endl;
      list1.readFromArrays(nameArray, phoneArray, SIZE);
      list1.print();

      // adding elements to list1 using insert and insertAtPos
      cout << “----------------------------------------” << endl;
      cout << “Adding records to list1 and printing list1” << endl;
      list1.insert(“Henry Hatch”, “617-227-5452”);
      list1.insert(“Kevin Etheredge”, “617-437-5454”);
      list1.insert(“Xavier Perez”, “352-654-1983”);
      list1.insertAtPos(“Adam James”, “202-872-1010”, 1);
      list1.insertAtPos(“Peter Anderson”, “352-654-2000”, 7);
      list1.insertAtPos(“Jamie Roberts”, “202-832-1560”, 3);
      list1.insertAtPos(“Nancy Garcia”, “617-227-5454”, 8);
      list1.insertAtPos(“Yara Mendiola”, “667-277-1454”, 11);
      list1.print();

      // printing list1 in reverse order
      cout << “----------------------------------------” << endl;
      cout << “Printing list1 backwards” << endl;
      list1.printReverse();

      // searching for specific entries in list1
      cout << “----------------------------------------” << endl;
      cout << “Searching for specific people in list1” << endl;
      list1.searchByName(“Nancy Garcia”);
      list1.searchByName(“Kevin Etheredge”);
      list1.searchByName(“Jamie Roberts”);
      list1.searchByName(“Julia Sarkis”);

      // updating specific entries in list1
      cout << “----------------------------------------” << endl;
      cout << “Updating records in list1 and printing list” << endl;
      list1.updateNumber(“Peter Anderson”, “989-876-1234”);
      list1.updateNumber(“Nancy Garcia”, “345-467-1234”);
      list1.updateNumber(“Jamie Roberts”, “202-447-1234”);
      list1.updateNumber(“Jamie Garcia”, “345-467-1224”);
      list1.print();

      // creating another list list2 and adding some elements to it
      cout << “----------------------------------------” << endl;
      cout << “Adding nodes to list2 and printing list 2” << endl;
```

```
    LL list2;

    list2.insertAtPos("Mary Mitchell", "617-227-5454", 1);
    list2.insert("Adam Sage", "202-857-1510");
    list2.print();

    // calling operator = to modify list2 so it is a copy of list1
    list2 = list1;

    // creating another list list3 using copy constructor
    LL list3(list1);

    // modifying list1 by removing some elements of list1
    list1.removeRecord("Mayssaa Najjar");
    list1.removeRecord("Adam James");
    list1.removeRecord("Yara Mendiola");
    list1.removeRecord("Jamie Najjar");

    // printing list1, list2, and list3 to make sure that a deep
    // copy is performed
    cout << "-----------------------------------------" << endl;
    cout << "Printing list1 after removing some records" << endl;
    list1.print();

    cout << "-----------------------------------------" << endl;
    cout << "Printing list2 after calling operator=" << endl;
    list2.print();

    cout << "-----------------------------------------" << endl;
    cout << "Printing list3" << endl;
    list3.print();

    // reversing the nodes in list3
    cout << "-----------------------------------------" << endl;
    cout << "Printing list3 after reversing it" << endl;
    list3.reverse();
    list3.print();

    // searching for a specific person in list3
    cout << "-----------------------------------------" << endl;
    cout << "Searching for specific person in list3" << endl;
    list3.searchByName("Jamie Roberts");

    return 0;
}
```

Once done, you need to submit the following:

1. A hard copy (printed copy) of the source codes (Node.cpp, LL.cpp, main.cpp and Node.h, LL.h) and output run placed in a folder. Use the script command to record your output. Start recording before you compile your code. You should hand in the folder during the first 5 minutes of the class when the homework is due.
2. An electronic copy of your source codes. You should place all source codes in a zipped folder and name it based on your first/last name and the assignment number. For example, JoeSmith-A1.zip. You should submit the zipped folder online through cougar website.