Programming Assignment 3: Linked List

Due: 11:59 pm the same day your lab meets the week of September 19th.

1. Introduction

The purpose of this lab is to provide you with an opportunity to use linked lists to manage simple sets of data. Specifically, your program should maintain a **doubly-linked** list of both doctors and patients of the KU Hospital. At the top level, there will be an ADT for the Hospital class, which will contain a doubly-linked list of Doctors. Additionally, there will be a second ADT for the Doctor class, which will contain a doubly-linked list of Patients.

Your program will process commands to *insert*, *delete*, and *search* Patients in the Doctors ADT linked list, as well as commands to *search* for a patient and to *display* either a single doctor's patient list or the entire patient list.

The starter code provides only a skeleton main.cpp file and a makefile. The makefile assumes that the classes describing the Hospital, Doctor, and Patient are each implemented as a separate pair of header and implementation files. The main.cpp file contains only a #include for “Hospital.h” because the example solution had “Hospital.h” include “Doctor.h” and “Doctor.h” included “Patient.h”. It is also reasonable to use a flat style of including files, where a given C++ source file includes any header files it needs directly. There are advantages and disadvantages to both approaches.

2. Getting Started

Your program will be expected to read input from the first file specified on the command line. Your program should display an error message and exit immediately if the expected command-line parameters are not provided. Following is an example of how your program will be run from the command line:

```
$ ./linkedList input.txt
```

Your program will implement at least three (3) classes: Hospital, Doctor, and Patient. You may or may not find that implementing additional classes helpful. The example solution did not.

Note that this description is meant to be a reasonable starting place but it does not describe every single component of the program you will have to implement. You will have to decide on many implementation details for yourself, using what you have learned about C++ previously in this class and others.

2.1. Hospital

This class will define an ADT containing the pointers and structures necessary to hold a linked list of Doctors. This will include both a Node structure defining the linked list node and a pointer to the head of the list. Doctors in this list will be stored in the order in which there are hired.

Note: Rather than repeating Doctor* throughout your ADT as the data type of your linked-list node item, you will need to create a new data type named NodeItemType using typedef. All references to the data type of your node item should use NodeItemType. This approach makes reading the linked list related code easier as it encapsulates exactly what data is managed by list nodes. This approach also makes writing C++ templates a bit easier.

Methods for this class should include:

addPatient(doctor_first, doctor_last, patient_object): When a new patient is added to a doctor's list, the object representing the doctor is searched for by first and last names in the liked list of doctors maintained by the Hospital object. If the doctor is not found, this routine should return an error message and the program should continue to the next command. Once the correct doctor is found, the new patient is added to the doctor's list by calling the appropriate member function of the Doctor class. The object representing the patient may be created in your main loop by overloading the input operator (>>) for the patient class to encapsulate the code that knows how to read sets of words from the input file to create a patient record..

deletePatient(doctor_first, doctor_last, patient_first, patient_last): When a patient is deleted from a doctor's list, the doctor list is searched for the appropriate doctor. If the doctor is not found, an error message should be displayed and the program should continue to the next command. If the correct doctor has been found, the patient is deleted from the event by calling the appropriate member function of the Doctor class.

searchPatient(doctor_first, doctor_last, patient_first, patient_last): When the set of patients of a doctor is to be searched, the doctor list is searched for the doctor. If the doctor is not found, an error message should be displayed. If the correct doctor has been found, search the list of patients for the patient calling the appropriate member function of the Doctor class.

hireDoctor(doctor_first, doctor_last): When a new doctor is hired, the new object representing the doctor is added to the end of the doctor list.

fireDoctor(doctor_first, doctor_last): When a doctor is fired, the doctor list is searched for the appropriate doctor. If the doctor is not found, an error message should be displayed. If the doctor has been found, the doctor will be removed from the doctor list. When a doctor is fired, we assumed that his/her patients will leave the hospital.

searchDoctor(doctor_first, doctor_last): The doctor list is searched for the appropriate doctor. If the doctor is not found, an error message should be displayed. If the correct doctor has been found, it will return the doctor pointer.

print: When the patients are to be printed for every doctor, the doctor list will be traversed. For each doctor in the list, the list of patients will be printed by calling the appropriate member function of the Doctor class. (**Note:** This function can also be implemented as an overloaded output operator.)

Here are some prototypes for the above member functions:

```
void addPatient(string doctorFirstname, string doctorLastname, Patient *patient);
void deletePatient(string doctorFirstname, string doctorLastname, string firstname, string lastname);
Patient* searchPatient(string doctorFirstname, string doctorLastname, string firstname, string
lastname);

void hireDoctor(string doctorFirstname, string doctorLastname);
void fireDoctor(string doctorFirstname, string doctorLastname);
void searchDoctor(string doctorFirstname, string doctorLastname);

friend ostream& operator<<(ostream &os, Hospital &hospital);
```

2.2. Doctor

This class will define an ADT containing the pointers and structures necessary to hold a linked-list of Patients. This will include both a Node structure defining the linked-list node for the list of patients and a pointer to the head of the list. Patients in this list will be sorted by last name alphabetically. This will require the insertion routine for the list to handle insertions any where in the list, not just the end.

Note: Rather than repeating Patient* throughout your ADT as the data type of your linked-list node item, you should create a new data type named NodeItemType using typedef. All references to the data type of your node item should use NodeItemType.

Methods for this class should include:

addPatient: When a new patient is added to a doctor's patient list, the patient list is searched for the appropriate position to insert the patient. The list is in alphabetical order.

```
void addPatient(Patient *patient);
```

deletePatient: When a patient is deleted from a doctor's patient list, the patient list is searched for the appropriate patient by first and last names. If the patient is not found, an error message is displayed and the program continues to the next command. If the correct patient has been found, the patient is deleted from the linked-list.

```
void deletePatient(string firstname, string lastname);
```

searchPatient: The patient list is searched for the appropriate patient. If the patient is not found, an error message is displayed and the program continues to the next command. If the correct patient has been found, the patient pointer is returned.

```
Patient* searchPatient(string firstname, string lastname);
```

print: When the doctor's patient list is to be printed, the patient list will be traversed. For each patient in the list, the patient name, age and ailment will be printed. The patient's information will be printed by calling the appropriate member function of the Patient class. Note that this function can also be implemented as an overloaded output operator. Note also that the code printing the information for the Doctor object would probably make use of the overloaded << operator for the Patient object as it printed the data for each patient on the Doctor's list.

```
friend ostream& operator<<(ostream &os, Doctor &doctor); //void Print(ostream &os);
```

2.3. Patient

This class will contain a patient's name, age and ailment.

Methods for this class should include:

print: This method should print the contestant's name, age and ailment. (**Note:** This function can also be implemented as an overloaded output operator.)

3. Input & Output

Your program will be expected to handle the following commands:

hire *fist-name last-name*

- This command will create a new doctor and insert at the end of the hospital's doctors list.

fire *fist-name last-name*

- This command will delete the doctor from the hospital's doctors list. It will search for the appropriate doctor. If not found, an error message will be displayed. If found, the doctor will be removed.

print-doctor *fist-name last-name*

- This command will print the patient list of the appropriate doctor. It will search for the doctor in the doctors list. If not found, an error message will be displayed.

add-patient *doctor-first-name doctor-last-name first-name last-name age ailment*

- This command will create a new patient and insert the patient at the correct position in the appropriate doctor's patient list. (Alphabetical order)

remove-patient *doctor-first-name doctor-last-name first-name last-name*

- This command will remove a patient from the appropriate doctor's patient list. If the doctor or the patient is not found, the command is ignored and an error message is displayed.

print-patient *doctor-first-name doctor-last-name first-name last-name*

- This command will print patient information from the appropriate patient in the doctor's patient list. If the doctor or the patient is not found, the command is ignored and an error message is displayed.

print-hospital

- This command will print the patient list of all the doctors and patients in the hospital.

3.1. Sample Input

```
hire John Smith
hire Jane Doe
hire Paul Newman
add-patient Paul Newman Ben Mitchell 23 cold
add-patient Paul Newman Oscar Lopez 30 cough
add-patient Paul Newman Gloria Zubizarreta 15 headache
add-patient Jane Doe Bo Lava 48 asthma
add-patient Jane Doe Diana Aldana 4 flu
add-patient John Smith Tony Seinfield 80 arthritis
print-hospital
remove-patient Paul Newman Oscar Lopez
print-hospital
fire John Smith
print-hospital
print-doctor Jane Doe
print-patient Paul Newman Ben Mitchell
```

3.2. Sample Output

Consider the example input file above. It begins with a set of commands that populate the model of the hospital with three doctors and a number of patients. Dr. Paul Newman 3 patients, Dr. Jane Doe has 2, and D. John Smith has only one. When these commands are complete, the “print-hospital” command states that the state of the hospital should be output. That produces the first block of output block:

```
------------------
Hospital: KU Hospital
------------------
List of doctors:
Doctor: John Smith
  List of patients:
  - Tony Seinfield, 80 - arthritis

Doctor: Jane Doe
  List of patients:
  - Diana Aldana, 4 - flu
  - Bo Lava, 48 - asthma

Doctor: Paul Newman
  List of patients:
  - Oscar Lopez, 30 - cough
   –  Ben Mitchell, 23 – cold
   –  Gloria Zubizarreta, 15 - headache
```

Then the example input command file removes the patient Oscar Lopez and again commands that the state of the hospital be output. That produes the second block of output:

```
------------------
Hospital: KU Hospital
------------------
List of doctors:
Doctor: John Smith
  List of patients:
  - Tony Seinfield, 80 - arthritis
```

```
Doctor: Jane Doe
  List of patients:
  - Diana Aldana, 4 - flu
  - Bo Lava, 48 - asthma

Doctor: Paul Newman
  List of patients:
  - Ben Mitchell, 23 - cold
  - Gloria Zubizarreta, 15 - headache
```

Then, the command to fire Dr. John Smith is executed, and the state of the hospital is produced a third time:

```
------------------
Hospital: KU Hospital
------------------
List of doctors:
Doctor: Jane Doe
  List of patients:
  - Diana Aldana, 4 - flu
  - Bo Lava, 48 - asthma

Doctor: Paul Newman
  List of patients:
  - Ben Mitchell, 23 - cold
   –  Gloria Zubizarreta, 15 - headache
```

Finally, the “print-doctor” and “print-patient” commands are executed producing:

```
Doctor: Jane Doe
  List of patients:
   –  Diana Aldana, 4 – flu
   –  Bo Lava, 48 - asthma

Ben Mitchell, 23 - cold
```

In general, your implementation should produce the same data if you execute it on the same input, although the formatting of the output can certainly be a bit different.

4. Grading criteria

50%	22 pts	Correctness of Class implementations
20%	9 pts	Correct main loop and I/O
15%	7 pts	Programming style and output formatting
15%	7 pts	Documentation
100%	45 pts	Total