CS 201 - Fall 2018

Homework Assignment 3

Due: 23:59, Dec 26, 2018

In this homework, you will implement a simple phonebook. A phonebook has a list of people. Each person has a name and a list of phone numbers. In your implementation, you **MUST** use linked lists. This homework will have two parts, whose requirements are explained below. This homework will be evaluated by ONUR KARAKASLAR (onur.karakaslar at bilkent edu tr). You may ask your homework related questions to him.

PART A:

To take the final exam, you MUST submit at least this part and MUST get at least half of its points.

This part is a simplified version of the entire system, where the user just creates the phonebook and enters the people only. So the system does not contain any phone numbers. In this system, you must keep the people in a linked list of the `Person` objects. Thus, you must implement the `Person` class first. This class is quite simple for Part A, but you will have to extend it for Part B.

1. Below is the required part of the `Person` class. The name of the class must be `Person`. The interface for the class must be written in a file called `SimplePerson.h` and its implementation must be written in a file called `SimplePerson.cpp`.
    - The `Person` class keeps the `name` of a single person as the sole data member. Make sure to implement the get function for this data member since we will use it to test your program.
    - Implement the default constructor, which initializes the `name` data member. Additionally, implement your own destructor and copy constructor, and overload the assignment operator. Although you may use the default ones for some of these special functions, you are advised to implement them (some may have no statements) so that it will be easier for you to extend them for Part B.
    - Do not delete or modify any part of the given data members or member functions. However, you may define additional functions and data members, if necessary.

```
#ifndef __SIMPLE_PERSON_H
#define __SIMPLE_PERSON_H

#include <string>
using namespace std;

class Person {
public:
   Person( const string name = "" );
   ~Person();
   Person ( const Person &personToCopy );
   void operator=( const Person &right );
   string getName();

private:
   string name;
};
#endif
```

2. Below is the required part of the `PhoneBook` class that you must write in Part A of this assignment. The name of the class must be `PhoneBook`. The interface for the class must be written in a file called `SimplePhoneBook.h` and its implementation must be written in a file called

SimplePhoneBook.cpp. Do not delete or modify any part of the given data members or member functions. You are not allowed to define additional functions and data members to this class for Part A.

```
#ifndef __SIMPLE_PHONEBOOK_H
#define __SIMPLE_PHONEBOOK_H

#include <string>
using namespace std;
#include "SimplePerson.h"

class PhoneBook{
public:
    PhoneBook();
    ~PhoneBook();
    PhoneBook (const PhoneBook& phoneBookToCopy);
    void operator=(const PhoneBook& right);
    bool addPerson(const string name);
    bool removePerson(const string name);
    void displayPeople();

private:
    struct PersonNode {
        Person t;
        PersonNode* next;
    };
    PersonNode *head;
    int numberOfPeople;

    PersonNode* findPerson(string name);
};
#endif
```

You must keep the recorded people in a linked-list of PersonNodes whose head pointer is PersonNode* head. In this class definition, you also see the prototype of a private function called findPerson. You may want to implement such an auxiliary function and use it in your add and remove functions (then in some other functions for Part B). This function takes the name of a person, searches it in the linked list of people, and returns a pointer to the PersonNode that contains that person, if the person exists in the system. Otherwise, it returns NULL. This auxiliary function may help you write more concise codes. However, if you do not want to use it, just define an empty function (with no statements) in your SimplePhoneBook.cpp file.

Things to do:

- Implement the default constructor, which creates an empty phonebook. Also overload the assignment operator and implement the destructor and copy constructors.
- Implement the add and remove person functions whose details are given below:

  Add a person: This function adds a person to the system. The name of the person is specified as a parameter. In this system, person names are unique. Thus, if the user attempts to add a person with an already existing name, do not add the person and return false. Otherwise, if the person does not exist in the system, add the person to the system and return true. DO NOT display any warning messages. Note that names are case insensitive (i.e., Machine Learning and MACHINE LEARNING are the same thing). You can't use any additional outside resources for comparing strings, you should only use standard libraries.

Remove a person: This function removes a person from the system. The name of the person to be deleted is specified as a parameter. If the person with the given name exists in the system, remove it from the system and return true. Otherwise, if there is no person with the given name, do not perform any action and return false. Likewise, DO NOT display any warning messages.

Display all people: This function should display the names of every person in the system one per line. If the there are no one in the system, display`--EMPTY--`.

```
Person name1
Person name2
. . .
```

3. To test Part A, write your own main function in a separate file. Do not forget to test your code for different cases such that the system is created and the above-mentioned functions are employed. However, do not submit this file. If any of your submitted files contains the main function, you may lose a considerable amount of points.

What to submit for Part A?

You should put your `SimplePerson.h`, `SimplePerson.cpp`, `SimplePhoneBook.h`and `SimplePhoneBook.cpp` files into a folder and zip the folder. In this zip file, there should not be any file containing the main function. The name of this zip file should be: PartA_secX_Firstname_Lastname_StudentID.zip where X is your section number. Then follow the steps that will be explained at the end of this document for the submission of Part A.

What to be careful about implementation and submission for Part A?

You have to read "notes about implementation" and "notes about submission" parts that will be given at the end of this document.

PART B:

In this part, you will extend the `PhoneBook` system you designed in Part A. For this, first, you are supposed to implement the `Phone` and `Person` classes whose interfaces are given below. Do not delete or modify any part of the given data members or member functions. However, you may define additional functions and data members, if necessary.

```
#ifndef __PHONE_H
#define __PHONE_H
using namespace std;

class Phone{
public:
    Phone();
    Phone( const int areaCode, const int number );
    int getAreaCode();
    int getNumber();

private:
    int areaCode;
    int number;
};
#endif
```

```
#ifndef __PERSON_H
#define __PERSON_H

#include <string>
using namespace std;

class Person{
public:
    Person( const string name = "" );
    ~Person();
    Person( const Person& personToCopy );
    void operator=( const Person &right );
    string getName();
    bool addPhone( const int areaCode, const int number );
    bool removePhone( const int areaCode, const int number );
    void displayPhoneNumbers();

private:
    struct PhoneNode {
        Phone p;
        PhoneNode* next;
    };
    PhoneNode *head;
    string name;
    PhoneNode* findPhone( const int areaCode, const int number );
};
#endif
```

Put the `Person` class in a file called `Person.h` and its implementation in `Person.cpp`, and put the `Phone` class in a file called `Phone.h` and its implementation in `Phone.cpp`. Implement all the functions given in the header files above. You must keep the phones of a person in a linked-list of `PhoneNodes` whose head pointer is `PhoneNode *head`. In this class definition of a `Person`, you also see the prototype of a private function called `findPhone`. You may want to implement such an auxiliary function and use it in your add and remove functions. This function takes the area code and the number of a phone, then searches it in the linked list of phones, and finally returns a pointer to the `PhoneNode` that contains that phone if the phone exists in the system. Otherwise, it returns NULL. This auxiliary function may help you write more concise codes. However, if you do not want to use it, just define an empty function (with no statements).

Here is some information about the functions to be implemented in the `Person` class:

Add a phone number: This function adds a phone number to the person. The area code and number of the phone are specified as parameters. In this system, phone numbers are uniquely identified by the area code and the number together. Thus, if the user attempts to enter a phone that exists for that person, display a warning message and return false. Otherwise, return true.

Remove a phone number: This function removes a phone from the person's record. The area code and the number of the phone to be deleted are specified as parameters. If there is no phone number in the list of the person, display a warning message and return false. Otherwise, return true.

Display phone numbers: This function lists all phone numbers already added to the person's record. The output should be in the following format. If the there are no phone numbers of the person, display `--EMPTY--`.

```
Phone number: Area Code1, Number1
Phone number: Area Code2, Number2
. . .
```

Then, extend the `Person` class from Part A, such that now it keeps the phone numbers of a person. These phones must be kept in another **LINKED-LIST**. Note that the number of phones for a person is not known in advance. Here, do not forget to implement the constructor, destructor, and copy constructor of this `Person` class as well as do not forget to overload its assignment operator. Otherwise, you may encounter some unexpected run-time errors. This time, the interface of the `Person` class must be written in a file called `Person.h`, and its implementation must be written in a file called `Person.cpp`.

After extending the `Person` class, now work on the implementation of the following functionalities that your PhoneBook system should support:

1. Add a person
2. Remove a person
3. Display all people
4. Add a phone to a person
5. Remove a phone from a person
6. Show detailed information about a particular person
7. Find the people associated with a specific area code

Add a person: This function adds a person to the system. The name of the person is specified as a parameter. In this function, the phone list is not specified; the phone(s) will be added later. In this system, person names are unique (case insensitive). Thus, if the user attempts to enter a person with an already registered name, display a warning message and return false. Otherwise, if the person is correctly added to the system, then return true. This function is very similar to what you will implement in Part A. But now, for Part B, you will need to create an empty phone list for the person when you add it to the system.

Remove a person: This function removes a person from the system. The name of this person is specified as a parameter. If there is no person with the given name, display a warning message and return false. Otherwise, if the person is correctly added to the system, then return true. This function is very similar to what you will implement in Part A. But now, for Part B, you will need to remove its phone list when you remove the person from the system.

Display all people: This function lists all people already registered in the system along with the number of phone numbers registered for them. The output should be in the following format. If the there are no people in the system, display `--EMPTY--`.

```
Person name1, number of phones name1 has
Person name2, number of phones name2 has
. . .
```

Add a phone to a person: This function adds a phone to the phone list of a person. The person name for which the phone is submitted to, is specified as a parameter. Also the area code and the phone number are parameters to this function. In this function, you should take care of the following issues:

- If the person with the specified name does not exist in the system, display a warning message and return false.
- All phone numbers are unique within the same phone list of a person. Thus, if the user attempts to add an existing phone to a person, display a warning message and return false. However, different people can have the same phone number. Note that a phone number can be shared among people (i.e., home phone number).

If above mentioned criteria are met, then the phone can be added to the person and this function returns true.

<u>Remove a phone from a person:</u> This function removes a phone from the phone list of a person. The person name for which the phone is to be deleted, and the area code and the number to be deleted are specified as parameters. If there is no person with the specified name or if the specified phone number (area code + number) is not in the phone list of the specified person, display a warning message and return false. Otherwise, return true.

<u>Show detailed information about a particular person:</u> This function displays all phone numbers of a person whose name is specified as a parameter. The output should be in the following format. If the person with the specified name does not exist in the system, display `--EMPTY--` after the first line.

```
Person name
Phone number: Area Code1, Number1
Phone number: Area Code2, Number2
Phone number: Area Code3, Number3
```

<u>Find the people associated with a specific area code:</u> This function lists all the people whose phone number lists contain a phone with the specified area code. The output should be in the following format. First write the queried area code. Then, list the names of the people who have a phone with the specified area code along with the phone number(s) of that person **with that area code only**. If a person does not have a phone number with that area code do not list him/her at all.

```
Area Code1
Person name (for the 1st person)
Phone number: Area Code1, Number1
Phone number: Area Code1, Number2

Person name (for the 2nd person)
Phone number: Area Code1, Number3
Phone number: Area Code1, Number4
. . .
```

If nobody in your system has a phone number with that area code, write `--EMPTY--` after the area code that is being searched for.

```
Area Code1
--EMPTY--
```

Below is the required public part of the `PhoneBook` class that you must write in Part B of this assignment. The name of the class must be `PhoneBook`. The interface for the class must be written in a file called `PhoneBook.h` and its implementation must be written in a file called `PhoneBook.cpp`. Your class definition should contain the following member functions and the specified data members. However, this time, if necessary, you may also define additional public and private member functions and data members in your class. You can also define additional classes in your solution. On the other hand, you are not allowed to delete any of the given functions or modify the prototype of any of these given functions.

```
#ifndef __PHONEBOOK_H
#define __PHONEBOOK_H

#include <string>
using namespace std;
#include "Person.h"
class PhoneBook {
public:
    PhoneBook();
    ~PhoneBook();
    PhoneBook( const PhoneBook& systemToCopy );
    void operator=( const PhoneBook &right );
    bool addPerson( string name );
    bool removePerson( string name );
    bool addPhone( string personName, int areaCode, int number );
    bool removePhone( string personName, int areaCode, int number );
    void displayPerson( string name );
    void displayAreaCode( int areaCode );
    void displayPeople();

private:
    struct Node {
         Person t;
         Node* next;
    };
    Node *head;
    int numberOfPeople;
    Node* findPerson( string name );
};
#endif
```

<u>What to submit for Part B?</u>

You should put your `Phone.h`, `Phone.cpp`, `Person.h`, `Person.cpp`, `PhoneBook.h`, and `PhoneBook.cpp` (and additional .h and .cpp files if you implement additional classes) into a folder and zip the folder. In this zip file, there should not be any file containing the main function. The name of this zip file should be: PartB_secX_Firstname_Lastname_StudentID.zip where X is your section number. Then follow the steps that will be explained at the end of this document for the submission of Part B.

<u>NOTES ABOUT IMPLEMENTATION (for both Part A and Part B):</u>

1. You MUST use **LINKED-LISTs** in your implementation. You will get no points if you use automatically allocated arrays, dynamically allocated arrays, or any other data structures such as `vector/array` from the standard library.
2. Do not delete or modify any part of the given data members or member functions for the given classes. You are not allowed to define additional functions and data members for classes in Part A, but you may do so for Part B, if necessary.
3. You ARE NOT ALLOWED to use any global variables or any global functions.
4. Your code must not have any memory leaks for Part B. You will lose points if you have memory leaks in your program even though the outputs of the operations are correct.
5. Your implementation should consider all names as case insensitive.

NOTES ABOUT SUBMISSION (for both Part A and Part B):

1. Conform to the rules given separately for Part A and Part B. That is, the name of the classes, the name of the .h and .cpp files, and the name of the zip files should conform to the specifications separately given for Part A and Part B. Otherwise, you may lose a considerable amount of points.
2. Before **23:59 on Dec26,** you need to send an email with a subject line "**CS201-HW3**" to **ONUR KARAKASLAR**(onur.karakaslar at bilkent edu tr), by attaching two zip files (one for Part A and the other for Part B). Read "what to submit for Part A" and "what to submit for Part B" sections very carefully. You may ask your homework related questions directly to him, as he will grade this homework.
3. No hard copy submission is needed. The standard rules about late homework submissions apply.
4. Do not submit any files containing the main function. We will write our own main function to test your implementations.
5. You are free to write your programs in any environment (you may use either Linux or Windows). On the other hand, we will test your programs on "`dijkstra.ug.bcc.bilkent.edu.tr`" and we will expect your programs to compile and run on the `dijkstra` machine. If you cannot get your programs to work properly on the `dijkstra` machine, you would lose a considerable amount of points. Therefore, we recommend you to make sure that your program compiles and properly works on "`dijkstra.ug.bcc.bilkent.edu.tr`" before submitting your assignment.