Programming Design, Spring 2015
Homework 9

Instructor: Ling-Chieh Kung
Department of Information Management
National Taiwan University

To submit your work, please upload two CPP files for Problems 1 and 2 to PDOGS at http://pdogs.ntu.im/judge/. Each student must submit her/his individual work. No hard copy. No late submission. The due time of this homework is ***8:00am, May 18, 2014***. Please answer in either English or Chinese.

Before you start, please read Chapter 10 of the textbook.[1] The TA who will prepare the solution for this homework is ***Willy Liao***.

Problem 1

(50 points) Recall that in Homework 6 we constructed a friend network. Let's use classes to give it a new implementation. In particular, we will work on the following class:

```
class Person
{
private:
  const int id;
  unsigned int friendCount;
  int* friendList;
public:
  Person(int id);
  ~Person();
  int getId() const;
  bool addFriend(int anId);
  bool unfriend(int anId);
};
```

The class `Person` has three private instance variables: `id` is the unique id of a person, `friendCount` is the number of friends this person has, and `friendList` is the dynamic array containing the IDs of this person's friends. Note that `friendList` include real friends and unconfirmed friends; two persons are really friends if both of them consider the other as a friend. In other words, as long as a person sends a friend request, the request receiver's ID is put into `friendList`. The array `friendList` actually contains friend requests, not real friends.

There are five public instance functions: a constructor `Person(int id)`, a destructor `~Person()`, an ID getter `int getId() const`, and two functions for adding and removing a person into and from the friend list. The function `bool addFriend(int anId)` takes a person's ID `anId` and checks whether `anId` is not equal to the caller's ID or any ID in the current friend list. If no, `anId` is added into `friendList` and true is returned; otherwise, nothing happens to `friendList` and false is returned. Similarly, the function `bool unfriend(int anId)` takes a person's ID `anId` and checks whether `anId` is equal to any ID in the current friend list. If yes, that ID is removed from the list and true is returned; otherwise, nothing happens to `friendList` and false is returned.

In this problem, you will be given a sequence of events in a network. There are three types of events:

- Event N: A new person joins the network. At that moment, she/he has no friend request.
- Event A: A person sends a friend request to another person in the network.
- Event R: A person unfriends another person in the network by removing the friend request.

[1]The textbook is *C++ How to Program: Late Objects Version* by Deitel and Deitel, seventh edition.

After your program processes these events, your program should report the IDs of those persons who have the most real friends.

Input/output formats

There are 15 input files. In each file, there are several lines. Each line start with a single English letter, which may be `N`, `A`, and `R`. These letters corresponds to the following events:

- If the first letter is `N`, it will be followed by a positive integer as the new comer's ID. You may assume that the ID is not the same as any existing person and is no greater than 1000. `N` and the ID is separated by a white space.
- If the first letter is `A`, it will be followed by two positive integers as the IDs of two persons. This means the person of the first ID sends a friend request to the person of the second ID. You may assume that these two IDs come from existing persons. However, the second ID may be the same as the first one or an existing ID in the friend list of the person of the first ID. `A` and the two IDs are separated by two white spaces.
- If the first letter is `R`, it will be followed by two positive integers as the IDs of two persons. This means the person of the first ID tries to remove the person of the second ID from her/his friend list. You may assume that these two IDs come from existing persons. However, the second ID may be the same as the first one or not an existing ID in the friend list of the person of the first ID. `R` and the two IDs are separated by two white spaces.

For example, for a file containing

```
N 1
N 2
N 3
N 4
A 3 2
A 1 3
A 3 4
R 1 3
A 2 3
A 3 1
```

as the input data, the network after all these events happen will be having four persons 1, 2, 3, and 4. Only persons 2 and 3 are real friends; person 3 has an unconfirmed request for person 4, and person 2 has an unconfirmed request for person 1. After processing these events, your program should output the IDs of the persons having the most real friends, from small to large, as well as that number of real friends. Two consecutive output values should be separated by a white space. For the above example, the output should be

```
2 3 1
```

in a single line. These three numbers are the IDs of the persons having the most real friends (2 and 3) and that number of real friends (1). Suppose the input file extends from the above to contain the following lines

```
A 3 2
A 3 3
R 1 1
R 2 4
```

there should be no change to the friend network. These kinds of invalid events may show up in the input files. Be careful!

What should be in your source file

You are required to implement a class `Person` by completing the function implementation for the class given in this problem. Then in your main function, you should use the class to complete the given task. If you need any member of `Person` that is not defined above, feel free to add it. However, your `Person` must have at least the members defined in this problem.

Your .cpp source file should contain C++ codes that will both read testing data and complete the above task. For this problem, you are NOT allowed to use techniques not covered in lectures. You should write relevant comments for your codes.

Grading criteria

5 points for this program will be based on your class `Person`. If it contains everything defined in this problem, you get 5 points.

30 points for this program will be based on the correctness of your output. PDOGS will compile your program, feed testing data into your program, and check the correctness of your outputs. For each set of input data, if your program outputs correctly without violating the space limit, you get 2 points. The 15 testing files are organized in the following way:

- In the first 10 files, there are only type-N and -A events.
- For the last 5 files, there are all three types of events.

15 points for this program will be based on how you write your program, including the logic and format. Please try to write a robust, efficient, and easy-to-read program.

Problem 2

(50 points) After you finish Problem 1, you should realize that the way to record the friend list is not very good. In particular, knowing whether a friend request has been confirmed is not easy. To see this, suppose that Person i has a friend request for Person j. This means j is included in Person i's friend list. Now we ask: Does Person j also include i in her/his friend list? To see this, we must first search for all `Person` object to see if any of them has its ID as j. Once we find it, we look at its friend list to see whether i is there. This can be very time consuming, especially when there are a lot of persons.

In this problem, we try another implementation of `Person` to make the process easier. The new definition of `Person` is

```
class Person
{
private:
  const int id;
  unsigned int friendCount;
  Person** friendList;
public:
  Person(int id);
  ~Person();
  int getId() const;
  void addFriend(Person* aPersonPtr);
  bool unfriend(int anId);
};
```

The main difference is that the friend list becomes a `Person` pointer array rather than an ID array. For example, if person 1 has sent friend requests to persons 4, 7, and 8, now `friendList` contains the addresses of the three objects for persons 4, 7, and 8, respectively. The instance function `addFriend` is also changed. Instead of adding an ID into the friend list, now the function add an address (of the objective of a person) into the friend list; that address is stored in the argument `aPersonPtr`.

With the new `friendList`, verifying whether a friend request has been confirmed is very simple. Consider `friendList[0]`, the first friend request. As it is the address of the object of a person, we immediately get the person's ID as `friendList[0]->getId()`. Moreover, we can look into `friendList[0]->friendList` to see if the friend request is confirmed. In general, if there are n persons in our system, we will have n objects; an object contains pointers pointing to an object if there is the corresponding friend request.

In this problem, you should revise `Person` to use the given definition. Then you will still be given a list of events for the same task. However, now there is the fourth type of event:

- Event L: A person leaves the network. At that moment, all the friend requests sending to her/him should be removed.

Removing all the friend requests sending to the one leaving the network may not be easy. Imagine how difficult it is if you are using the class defined in Problem 1! With the new definition, it is much easier.

Input/output formats

There are 15 input files. In each file, there are several lines. Each line start with a single English letter, which may be `N`, `A`, `R`, or `L`. These letters corresponds to the four events, where the first three are defined in Problem 1. The last one is defined below:

- If the first letter is `L`, it will be followed by a positive integer as the leaving person's ID. You may assume that the ID is owned by one existing person. `L` and the ID is separated by a white space.

When a person leaves, all friend requests sent to her/him must also be removed. Otherwise an error may occur. For example, for a file containing

```
N 1
N 2
N 3
N 4
A 3 2
A 4 2
L 2
N 2
A 2 3
```

as the input data, the number of confirmed friend request is 0, not 1! When person 2 leaves the network, the requests sent to her/him from persons 3 and 4 should be removed. Then when person 2 later comes back (with the same ID) and sends a request to person 3, that request is not confirmed at the end of the event list. Your program should output

```
1 2 3 4 0
```

where the first four numbers are all the IDs and 0 is the number of real friends each of them has.

What should be in your source file

You are required to implement a class `Person` by completing the function implementation for the class given in this problem. Then in your main function, you should use the class to complete the given task.

If you need any member of `Person` that is not defined above, feel free to add it. However, your `Person` must have at least the members defined in this problem.

Your .cpp source file should contain C++ codes that will both read testing data and complete the above task. For this problem, you are NOT allowed to use techniques not covered in lectures. You should write relevant comments for your codes.

Grading criteria

5 points for this program will be based on your class `Person`. If it contains everything defined in this problem, you get 5 points.

30 points for this program will be based on the correctness of your output. PDOGS will compile your program, feed testing data into your program, and check the correctness of your outputs. For each set of input data, if your program outputs correctly without violating the space limit, you get 2 points. The 15 testing files are organized in the following way:

- In the first 5 files, there are only type-N, -A, and -R events.
- For the last 10 files, there are all four types of events.

15 points for this program will be based on how you write your program, including the logic and format. Please try to write a robust, efficient, and easy-to-read program.