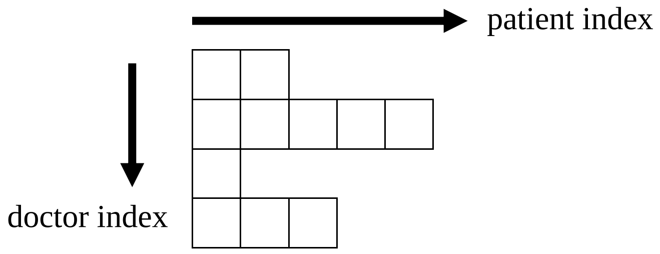
PA #5 – Part 4

***Problem:* Patient Management System**
***Due Date:* Beginning of Class, Friday, April 7, 2017**

The solution to the following problem must be completed by you and only you. You may seek help only from the professor and the tutor, and you may use your texts, notes, online tutorials, etc., but the code must be your own. Bear in mind that the professor and the tutor are helping you with your C++ problem and not writing your program! If you have trouble compiling or debugging your code, see the professor or the tutor as soon as possible.

Problem Statement

Using the data structure created from part 1 & 2, create a part of the front-end application for a clinic[1] for patient information management. The patients' information is store in a 2-dimension dynamic array, which can be visualized as follows where the size for each of the row does not have to be the same. The size of the row depends on the number of patient the doctor has.



The information on the doctor is stored in a file named ***doctors.txt***, and its format is given as follows:

Format	Example

`Number of Doctor`
`Doctor's Name`[2]
`Doctor's ID`[3]
`Doctor's Index`[4] `Number of Patient for the Doctor`
	`2`
	`John Doe`
	`123-45-6789`
	`0 5`
	`Jane Doe`
	`222-33-4444`
	`1 2`

1 You may name the facility anything you like, but nothing offensive.
2 You can assume that the doctor's name is unique.
3 You can assume that doctor's SSN is unique and it is used as his/her ID.
4 The index for the doctor in the array.

And each of the doctors has his/her own file to store the records for his/her patients, and this file is named using the doctor's SSN. For example, if Dr. John Doe's SSN is *123-45-6789*, then the name of his file is ***123-45-6789.txt***. The format for this file is given as below:

Format	Example
`Patient's Name`[5]	`John Smith`
`Patient's SSN`[6]	`000-00-0001`
`Patient's Address`	`123 West Ave, Burlington, VT 05401`
`Patient's Phone Number`	`(802) 865 0000`
`Doctor's ID for the Patient`	`123-45-6789`

Your system must allow the user to perform the following operations:

1. Add a new patient to the system,
2. Remove an existing patient from the system,
3. Search for a patient using an ID and
4. Update patient's name.

Implementation Details

1. All operations for patient must be stored in ***doctorOperations.cpp*** and the function prototypes must be declared in ***doctorOperations.h***. The following are the required functions for doctor, and you are welcome to add more functions if needed. Sample function prototypes are given to you as a suggestion, and you are more than welcome to adopt it, modify it or completely ignore it.

   a. For the given doctor name, return his/her index in the array
   ```
   int getDoctorIndex(Doctor doctors[], int numberOfDoctor, string doctorName);
   ```

   b. Determine if the doctor's name exist in the system
   ```
   bool isDoctorExist(Doctor doctors[], int numberOfDoctor, string doctorName);
   ```

   c. Load the doctor's information from ***doctor.txt*** into the doctor array. You will need to create the dynamic array for doctors within this function based on the size
   ```
   int loadDoctor(Doctor *&doctors);
   ```

   d. Store the doctor's information from the system back to the ***doctor.txt***
   ```
   void storeDoctor(Doctor doctors[], int numberOfDoctor);
   ```

   e. This is an anchor point for the doctor operations where you can add, remove, search and update doctor. This is a 20 points bonus, and you are not required to do this part.
   ```
   void doctorOperations(Patient **&patients, Doctor doctors[],
                         int numberOfDoctor);
   ```

[5] You cannot assume that the patient's name is unique.
[6] You can assume that the patient's SSN is unique and it is used as the ID for the patient.

陳

2. All operations for patient must be stored in ***patientOperations.cpp*** and the function prototypes must be declared in ***patientOperations.h***. The following are the required functions for patient, and you are welcome to add more functions if needed. Sample function prototypes are given to you as a suggestion, and you are more than welcome to adopt it, modify it or completely ignore it.

   a. This is the anchor point for the patient operations. Within this function, you should determine what the user wanted to do and call upon the corresponding function to carry out the task.

   ```
   void patientOperations(Patient **&patients, Doctor doctors[],
                          int numberOfDoctor);
   ```

   b. Add a new patient onto the system

   ```
   void addPatient(Patient **&patients, Doctor doctors[], int numberOfDoctor);
   ```

   - Prompt the user for all information for the new patient
   - Your system need to make sure that each patient is unique meaning not doctor can share the same patient. The name is not a good key to determine if the patient is unique or not because two distinct individuals may have the same name. However, two distinct individual cannot have the same SSN, at least we hope so. Therefore, use the SSN as a unique key to make sure each patient is seeing only one doctor
   - Prompt the name for the doctor of the new patient
   - Make sure that the doctor is accepting new patient

   c. Search for a patient information

   ```
   void searchPatient(Patient **patients, Doctor doctors[], int numberOfDoctor);
   void getPatientIndex(Patient **patients, Doctor doctors[], int numberOfDoctor,
                        string id, int &patientIndex, int &doctorIndex);
   ```

   - Prompt the user for a patient's SSN. If the SSN exists in the system, then display the information to the screen
   - You need to display the name of the doctor for the searching patient as well

   d. Determine if a given patient already existed in the system or not

   ```
   bool isPatientExist(Patient **patients, Doctor doctors[], int numberOfDoctor,
                       string id);
   ```

   e. Load the patients from the data file onto the system and place in the corresponding indexes for their doctor

   ```
   void loadPatient(Patient *&patients, Doctor doctor);
   ```

   f. Remove a patient from a doctor

   ```
   void removePatient(Patient **&patients, Doctor doctors[], int numberOfDoctor);
   ```

   g. Store the patient's information to the data file corresponding to their doctor

   ```
   void storePatient(Patient patients[], Doctor doctors);
   ```

   h. Ask the user for the ID for the current patient and update the patient with the latest name for the patient

   ```
   void updatePatient(Patient **patients, Doctor doctors[], int numberOfDoctor);
   ```

陳

3. You should defined and declared in ***commonOperations.h*** and ***commonOperation.cpp***, respectively, so that you can share these functions across the system. If you need more function that are share across all parts of the system, you can define and declare them here.

   a. To clear the console screen
      ```
      void clearScreen();
      ```

   b. To pause the execution flow until the user hit a key on the keyboard
      ```
      void pause();
      ```

Additional Requirements

1. Your programming style must comply with the Programming Standard.
2. Your program must be compliable and executable according to the specification.
3. All files must bear header information.
4. Constants must be used.
5. Your functions must be sort alphabetically.
6. Each of your function must have the following information:
   a. Purpose of the function,
   b. Pre-condition and
   c. Post-condition.
7. You must use blank lines to separate functions so that they can be read easier.
8. No global variable is allowed.
9. You must have a stub driver to test your function and you should name your main program as ***pa5-part4.cpp***.
10. If you need to declare more function, declare and define these functions in ***header.h*** and ***functions.cpp***, respectively.
11. You must load all data from the files onto the arrays when the system starts.
12. You must store all data from the arrays back to the data files when exiting the system.