BİL 102 – Computer Programming
Project 02
Polyclinic Automation

Last Submission Date of the Analysis Report: June. 01, 2012 – 23:45

Last Submission Date of the Implementation: June. 04, 2012 – 23:45

1. Description of the Project

In this project you will implement a simple polyclinic automation system working on the following basis:

- *The polyclinic has some clinics such as internal medicine, neurology, dermatology, radiology, etc.*
- *Doctors are the only personnel considered by the program. They work in a specific clinic.*
- *Patients are examined in a clinic. To be examined, a patient must make reservation for a clinic at last the previous day of the examination.*
- *For each clinic, define some working hours and a fixed period for each examination and taking the number of doctors in that clinic into account, determine the daily examination capacity of the clinic. Do not let any other reservations to a clinic if it has not any free capacity.*
- *All the input data will be available in the first run of your program (i.e., they will be saved into corresponding files). In the first run the data will be taken from the computer user.*
- *Make logical assumptions when required.*
- *For simplicity:*
    - do not consider pricing issues in your implementation
    - you may allow reservation and medical examination in all days including weekends and other holidays,
    - you can organize the reservations as follows:
        - let the user specify **the day only**
        - if there exists some free capacity of the clinic for that day
            - determine a suitable examination time (simply choose the first available time slot) and a suitable doctor
            - realize the reservation, inform the user
        - else
            - reject the reservation demand
    - do not consider the current time, consider only the current date; for a specific date, realize the examinations in the reservation order one by one and assume that the time of the examination is the same as the time of the reservation for records,
    - No need for deleting a clinic (because it would require to handle some complications)

- you can make some other **reasonable** simplifications in details to avoid massively large codes.

- *Your implementation should have at least the following capabilities from the perspective of the user*:
  - Adding a new clinic.
  - Adding a new/deleting or modifying an existing doctor.
  - Adding a new/modifying an existing patient.
  - Setting Current Date
  - Adding a new/deleting or modifying an existing reservation.
  - Realizing a medical examination (recording outputs of the examination like diagnosis, treatment and etc).
  - Monitoring the following information (You may add more capabilities):
    - Clinics and their capacities, all patients examined in a given day in a given clinic, brief examination outputs for each, etc.
    - Doctors in any clinic
    - Information of any doctor (given his/her name): all information about him/her
    - Information of any patient (given his/her name): all information about him/her such as personal information (communication information, age, etc.), medical history he/she declared, list of all medical examinations done in the polyclinic, list of all future reservations.
  - Generating the following reports in text files (You may add more):
    - Detailed polyclinic's information: includes the clinics it has, all information about each clinic (doctors, capacity, working hours, etc.)
    - Detailed personnel information: lists all doctors with all information about each one.
    - Detailed clinic's information: includes all information about the clinic, all of its doctors, and all of the information about each doctor.
    - Detailed information of all patients: lists all patients and all information about each.
- *In each run, your program should try to load the information from some binary files (that we explain below) recorded in the last run. If such files do not exist (i.e. this is the first run), it should ask the minimum information from the user to build a valid polyclinic model and create the files before making anything else.*

2. Some Design, Implementation and Testing Issues

The main purpose of this project is to provide you an opportunity to practice in basic data structures and files. Therefore, representing data is very important. You should carefully analyze the requirements and design your types of variables including the following ones:

- A 'human' structure to hold basic information of any person
- Some types of variables to represent all actors in the polyclinic (patients, doctors, …), they should include human as a member and have another member 'id' to define each actor in its genre uniquely (and certainly some other members)
- A date and a time variable type.
- Some types of variables to represent any reservation (holding at least a unique reservation id, patient's id, doctor's id, clinic's id, date, time) and any medical examination
- A type to hold the information of any clinic

Your data representation repertoire should include **at least one union.**

Your program should generate **binary files with 'bin' extension to remember the given information** in the next run and **text files with 'txt' extension to demonstrate some reports**. Binary files should include (but certainly not limited by) the following ones:

- "clinics.bin" to hold the information of the clinics
- "doctors.bin" to hold all the information of all the doctors (do not use any other files to represent the doctors of a clinic)
- …

You can use a hex editor for observing/creating/manipulating binary files for testing, debugging or tracking purposes. Notice that, still you may have problems in interpreting variables.

We note the following issues:

- Even after a doctor is deleted by the user, past records referencing that doctor should be able to be accessed and monitored properly, e.g., assume that a doctor is fired, therefore, the user commands your program to delete that doctor. Even after this, examination list of a patient, who was examined by this doctor, should be monitored properly (with correct doctor name). But, that doctor should not be considered for the future reservations for example. In such cases information should not really be deleted. Instead, you can use a binary variable which indicates if the record is active (or if it is kept only for monitoring information history properly).
- In the text book, it is shown how to open a file for reading and creating a file for writing. But, it is also possible and required in this project to add records to an existing file without deleting past records. Search the function "fopen", investigate its parameters and learn how to do this.

After deciding how to represent information, you should determine how to manipulate it. Determine the functions and their arguments so that **at least 5 of them have an argument of a user-defined type's pointer.** Write a "polyclinicAutomation" function which executes your program when called by "main". Produce 3 coding files for your implementation:

PR02_<student number>_Main.c

PR02_<student number>_Polyclinic.c

PR02_<student number>_Polyclinic.h

In the first file, there should be main function and any other helper functions of main, and the prototypes of these helper functions only. The second file includes the implementation of all the functions of the project, including the "polyclinicAutomation" function. And the third file, the header file, includes all the function prototypes of the project. When compiling, all files should be in the same folder, the first and the second file should include the third file by using quotation mark instead of '<' and '>' marks (which indicates that the file to be included will be searched in the current folder). You should compile the first and the second files and link using the generated two object files, as you did in "Cin Ali" homework.

Search conditional preprocessor directives (#ifdef, #ifndef, #if, #endif, #else and #elif) and use them:

- in the header file to provide it to be compiled only once (refer to CinAli homework), otherwise the compiler should produce an error,
- for defining 2 working modes for the system: debug and normal.

In debug mode, printout some critical variables to track the states of the system. In normal mode, do not make any unnecessary printouts.

3. BONUS (%50)

The more capabilities your code has, the more realistic it is and the less simplification you make mean the more bonus points you will get. Meanwhile, do not insert unnecessary complexity to your code. For bonus points, your program must be very user friendly and it must have proper structures having enough number of components to represent more information that may be needed for better reports. Also you should have enough number of functions to implement abstract data types efficiently. You may add more functionality like collecting statistical data about the doctors, patients or clinics.

Another source for bonus points is that: Provide a test mode using conditional preprocessor directives, in which all console outputs and inputs (except those used in debug mode) are directed to some text files. For this purpose replace console I/O functions with their stream processing versions (e,g, printf -> fprintf), using the file pointers "stdout" and "stdin" when the test mode is passive and some suitable file pointers when it is active. Also provide some suitable input files and test scripts (run only in test mode) to test your implementation.

Outputs of the Project

You will submit 3 documents, all of which are prepared in pdf format, 3 coding files and **several text files (only for the bonus part)** for this project:

Documents

Analysis report

This report will include:

- a detailed description of the project: write with your own words how it will work, what restrictions it has **in details**
- I/O report: define all inputs and outputs of your implementation
- a structure chart for the project (refer to chapter 4.5 of the text book)
- function prototypes: define the functions to be used in your implementation, write their prototypes, for each function give brief explanations about the function itself, each input of it, its output and the functions it calls (also mention which functions polyclinicAutomation function calls).

Implementation report

This report explains what you could realize. You should compare your result with your goals in the analysis report. Append your code to this document.

We note that, you will obtain higher credit if you mention any unimplemented requirements in the analysis report and discuss them here than the case in which these parts are not considered at all.

Test report

Make some tests and provide some outputs (screenshots of console and hex editors, contents of text files, etc.) of your program for some given inputs demonstrating that your program works properly.

If you also implement the bonus part, explain what you did for the bonus and provide the input and the output text files.

Coding Files

You should summit the following coding files:

PR02_<student number>_Main.c

PR02_<student number>_Polyclinic.c

PR02_<student number>_Polyclinic.h

Text Files

Send the input files you produce for the bonus part. **DO NOT send any other text files. All binary files used by your program should be created by your program in the first run by asking the user. In the subsequent runs, the program will use these binary files and it should be able to modify them whenever necessary.**

General:

1. Obey honor code principles.
2. Obey coding convention.
3. Deliver the printout of your code until 3 days later then the last submission date.
4. The most important part of this project is the Analysis part. Therefore, pay attention to decide the necessary abstract data types and the functions accompanying them.
5. The code communicates with the user through screen and keyboard. Therefore, your program should have a properly constructed interface (menu) for enabling the first time database construction and the functionalities explained in Part 1 under the bullet "*Your implementation should have at least the following capabilities from the perspective of the user"*.
6. These questions will help you in the design process:
   - Which *structures* do I need?
   - Which *components* will each structure have?
   - What *functions* do I need to implement to be able to use abstract data types?
   - What will the user be able to do? And how will he/she do that?
   - Who am I? What am I doing here? ;-))
   - …