CIS 15 Spring 2009, Assignment V

Instructions

- This is the assignment for Unit V.
- It is worth 10 points.
- **It is due on Wednesday May 6** and must be submitted by email (as below).
- **Follow these emailing instructions:**
  1. Create a mail message addressed to **parsons@sci.brooklyn.cuny.edu** with the subject line **cis15 hw5**.
  2. Write your name, that is the name under which you registered for the course, in the email. When I get an email from `deathmetal@aol.com` or `pinkprincess@yahoo.com`, I can usually guess whose program it is, but that is not as good as *knowing* whose program it is.
  3. Attach ONLY the files I tell you to send me (see below).
  4. Failure to follow these instructions will result in points being taken away from your grade. The number of points will be in proportion to the extent to which you did not follow instructions . . . (which can make it a lot harder for me to grade your work)

Program description

- For this assignment you will write a small database, more specifically a medical records database.
- As in the last couple of assignments, this will involve writing a number of classes, as well as a main program that makes use of the classes.
- Unlike the last few assignments, I haven't specified exactly what each class should contain, and how each function should work. I figure that by now you should be able to work this stuff out for yourselves. When I mark the homework, you will get more credit the closer your code is to the programming style that I have used in the sample code I have given you, the same style that the last few homeworks have spelled out in detail.
- For full credit, you should submit separate `.h` and `.cpp` for each class and a file **hw5.cpp** that contains the `main()`. However, I will also accept solutions that have everyting lumped together in one file (though you will get less credit).

A. The "person" class

The basic building block of the data in the database is a class `person`.

1. Create a `person` class. The class should have a `string` data members called $firstName$ and $lastName$ and an `int` data member called age.

   The class should have four function members:

   - A function to set the first name.
   - A function to set the last name.
   - A function to set the age.
   - A function to print out the data in an object of type `person`.
2. Create a `main()` for testing that creates an object `myPerson` of the `person` class and tests the functions.

   This `main` is for unit testing, you don't hand it in.

B. The “patient” class

The `patient` class records data that is specific to a patient.

1. Create a `patient` class that is a sub-class of `person`. The class should have a `string` data member called *disease* and an `int` data member called *id*.

   The class should have three function members:

   - A function to set the disease.
   - A function to set the id.
   - A function to print out the data in an object of type `patient`.

2. Create a `main()` for testing that creates an object `myPatient` of the `patient` class and tests the functions.

   This `main` is for unit testing, you don’t hand it in.

C. The “doctor” class

The `doctor` class records data that is specific to a doctor.

1. Create a `doctor` class that is a sub-class of `person`. The class should have a `string` data member called *specialty* and an `int` data member called *ssn*.

   The class should have three function members:

   - A function to set the specialty.
   - A function to set the ssn.
   - A function to print out the data in an object of type `doctor`.

2. Create a `main()` for testing that creates an object `myDoctor` of the `doctor` class and tests the functions.

   This `main` is for unit testing, you don’t hand it in.

D. The “record” class

The `record` class is an indirect way to store information about patients and doctors in the same datastructure.

1. Create a `record` class. The class should have an `int` data member called *id* and a data member called *pPtr* that is a pointer to `person`.

   The class should have four function members:

   - A constructor which sets the *id* to a unique value (you will likely need to use some global variable to achieve this).
   - A function to set the value of the pointer *pPtr*.
   - A function to return the value of *id*.
   - A function to return the value of *pPtr*.

2. Create a `main()` for testing that creates an object `myRecord` of the `record` class and tests the functions. Test the function that sets the pointer by making the record point to a patient object and then to a doctor object.

   This `main` is for unit testing, you don’t hand it in.

E. The "main" program

Now create a `main` that you submit. This should:

1. Ask the user how many records they want to store.
2. Use `new` to create an array that has as many records as the user says to create.
3. Include a function $addPatient$ that allows the user to add a `patient` object to the database.
4. Include a function $addDoctor$ that allows the user to add a `doctor` object to the database.
5. Include a function $lookup$ that allows the user to lookup either and print out a patient or a doctor in the database.

   You can do this by having the user enter the name, or the id (for a patient), or the ssn (for the doctor), or the record id (you choose which way to do this).
6. Include a function $printDB$ that allows the user to print out data about each person in the database (it is okay to just print out the name and age).

F. Submission

If you wrote your program as separate `.cpp` and `.h` files, use ZIP to create a single file **hw5.zip** and send this to me.
If you wrote your program as a single file, send that to me.

G. Marking rubric

This assignment is worth 10 points. The breakdown is as follows:

- `person` class with three data members and four function members.

  (1 point)
- `patient` class with two data members and three function members.

  (1 point)
- `doctor` class with two data members and three function members.

  (1 point)
- `record` class with two data members, a constructor, and three function members.

  (2 points)
- `main()` which allows the user to be able to specify the size of the database, which dynamically creates an array of `record` objects and which offers a menu of options.

  (2 points)
- The functions $addPatient$, $addDoctor$, $printDB$ and $lookup$ as specified above.

  (2 points)
- If the program is structured with separate `.h` and `.cpp` files for each class.

  (1 point)
- A `make` file that builds the whole homework.

  (1 extra credit point)