CS 2255 LAB04 (Continuation of Lab03) (Read all instructions)

LAB04 (Part 1)

The idea behind Lab04 is to bring Lab03 into the modern world of Object Oriented Programming. We have a simple class, C_Person, which is a start, but it has no methods and all attributes are public which is not always the best choice.

Part one of Lab04 is redesign the C_Person class where it does some of the work (processing) for us. Looking back at Lab03, we notice that we have several constructs that have to reach into the C_Person container and deal with each attribute individually. That requires several lines of code for each construct and it would be better if the class did some of the work for the calling function (a.k.a. main).

Create a method in the C_Person class that will display itself and the last date it was modified (if ask). Another word, given the construct that we used in Lab03:

```
for(int index = 0; index < V_People.size() - 1; index++)
{
            cout << V_People[index].lastName << ", "
                 << V_People[index].firstName << endl;
            cout << V_People[index].street << endl;
            cout << V_People[index].city << ", "
                 << V_People[index].state << " "
                 << V_People[index].zipCode << endl;
}
```

If we remove the "orange" text: remove the for loop and the reference to the vector, we end up with:

```
            cout << lastName << ", "
                 << firstName << endl;
            cout << street << endl;
            cout << city << ", "
                 << state << " "
                 << zipCode << endl;
```

If we create a C_Person class method with this new code construct, we can ask any C_Person to display itself to the console screen. We can then modify the same method to either output itself to the console screen or to an output file, depending on how the calling function ask (i.e., method overloading).

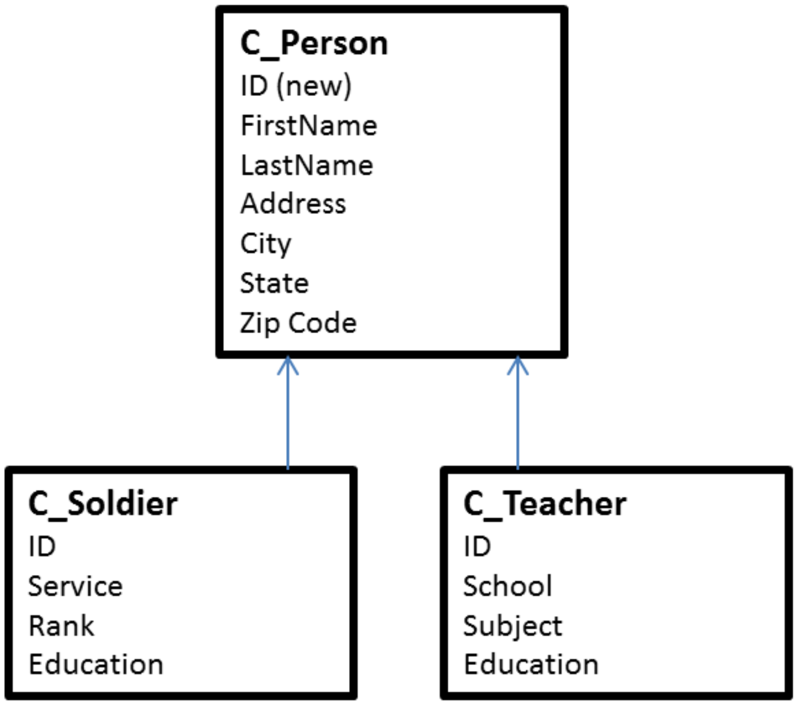
In addition, you will need to add a **date** class (we will cover that in class) and output it where-ever, when ask, in the proper format, when ask.

For Lab04, the main program will continue to "own" the vector and perform the File_IO, but we will treat the C_Person class different.

LAB04 (Part 2)

Using the C_Person class as our base class, we will design and implement two derived class blueprints for "C_Soldier" and "C_Teacher". Although we could have many, many more classes that could be derived from C_Person (i.e., Lumberjack, Clown, etc.), for our purpose, two will be enough to model the inheritance and polymorphic properties required for Lab04.

I have added an additional data field (an ID attribute) to the original Lab2Names.txt file in order to retrieve/match data items from a second input file. The second input file contain data fields related to either the derived C_Soldier class blueprint or the C_Teacher blueprint. We will diagram the base->dervived classes in class.



Note that you will need to make a decision as to whether attributes should reside in the base class or derived class. Since we have been using the C_Person class for previous Lab Assignments, Name and Address should remain in the base class. If is up to keep the class design "as-is" or adjust as needed (**you will be graded on this design decision**).

The new input files are: **Lab4Input1.txt** and **Lab4Input2.txt**

Your output file should be named: **Lab4OutNames.txt**

I will provide you with the files.

Your program will need to read in the data from both input files, properly construct the base-derived classes, store the objects onto one single vector of C_People, and output the data from the single vector to its respective output file: Lab4SoldiersOut.txt for all C_Soldiers and Lab4TeachersOut.txt for all C_Teachers.

You should create separate files for each class: Person.cpp, Person.h, Teacher.cpp, Teach.h, Soldier.cpp, and Soldier.h and #include those class blueprints/files where needed.

You were given Date.cpp and Date.h as examples of how these files should be laid out.
The major “new” subject matter related to this lab are:

1. Separate Files for classes, one header file to contain the blue print and one class implementation file (i.e., .cpp) file.
2. Inheritance: Base Class with two Derived Classes (Parent-Child relationship)
3. Polymorphism: Using a Vector to store all Derived Class Objects as Base Class Object Pointers
4. Virtual Functions (methods): Grab anything off the Vector of Base class pointers and tell it to execute a virtual function which has a version for one derived class and another for the other derived class. The correct one will execute. NEED TO USE POINTERS TO THE DERIVED OBJECTS MAKE THIS WORK.
5. Encapsulation: Have the class do the class related work. Display “itself” to Screen, Write “itself” to a File Stream.

Given the two new input files, the output file generated by your code should look like:

```
SSgt Tommy Gunn, USAF
22 Winchester Drive
Calamar, AL 38880
AS Information Technology

Bobby Smither
117 Drover Drive
Budville, NC 28711
Budville High School
Math
BS Math

William Hammet
45 East West Street
Springfield, NC 28881
Springfield Junior High
History
MA History

Capt Shenna North, USAF
10 Tenth Street
Canopener, AL 31110
BS Engineering

MSgt Robert Stillskin, USAF
300 Square Circle
Picklestown, NC 27887
BS Information Technology

Walter Mellon
99 Main Street
```

Blueville, AL 33990
Blueville High School
English
BA English

Tammy Smith
100 Washburn Street
Calamar, AL 38880
Washburn Elementary School
General Studies
BA Elemntary Education

TSgt Roger Sutton, USAF
55 East Cantor Circle
Budville, NC 28711
HS Diploma

Tamika Gailborn
8822 Eastern Avenue
Springfield, NC 28881
Springfield Junior High
English
BA English

Maj Richard North, USAF
1011 Ninth Street
Canopener, AL 31110
MA History