CSE 1384 - Classes and Inheritance
Lab 6

Objectives:

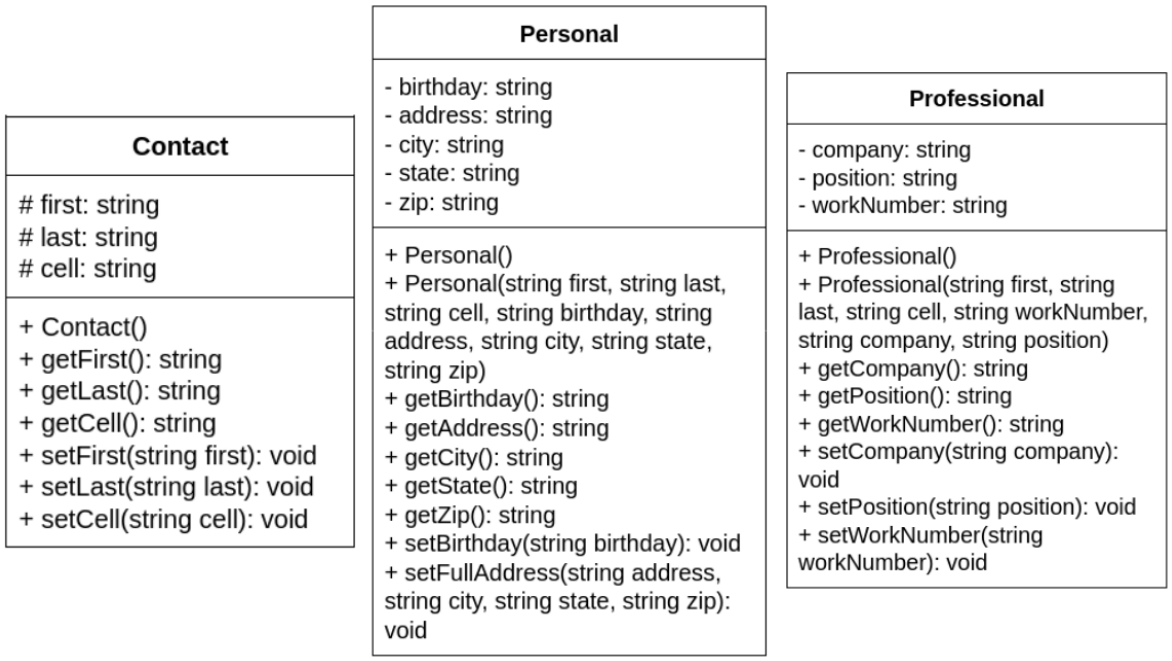
- Continue practicing past concepts
- Practice building basic C++ classes
- Practice building C++ classes that use inheritance

Assignment:

Build the three classes defined below:

- Contact (base class)
- Professional (derived class)
- Personal (derived class)

The UML diagrams for the classes are as follows:



The UML cheat sheet given in lab 4 is attached in this document as well.

For these classes, all getters/setters are standard getters/setters, except the setFullAddress function in the Personal class. This should set all variables: address, city, state, and zip. Personal and Professional both should have zero constructors that merely zero out all the values. They also have constructors with parameters that should set their variables and the variables of Contact.

Once you think you're done, test the classes against the test file provided for you: `test.cpp`. This has a concrete output that should match the following output exactly:

```
Personal 1:
John Lennon
Cell: 555-555-5555
Address:
A Place
Actually England, EN 12345
Birthday: October 9, 1940

Personal 2:
Ringo Star
Cell: 456-123-7890
Address:
Something Road
Certainly the UK, UK 42572
Birthday: July 7, 1940

Professional 1:
Paul McCartney
Cell: 444-444-4444
Work: 123-456-7890
Company: Wings
Position: Vocals

Professional 2:
George Harrison
Cell: 789-654-4321
Work: 901-345-9521
Company: The Beatles
Position: Guitarist
```

This should let you know that you have followed the UML diagrams correctly in your setup.

Once you've verified your classes work as intended, create your own main file. Your main function should first establish two vectors: one of Professional type, one of Personal type. Then, you should take in a file (you should verify this file exists in a loop) and open it for reading (`contacts.txt` has been provided to you).

The file format should have each contact indicated by "personal" or "professional". Then, you'll read from the file a certain number of times. Each field (i.e., first name, last name, cell number, etc) will be on separate lines. Take a look at the file given to see the order of the fields. This means, NOT including the personal/professional tags, personal contacts will have 8 lines and professional contacts will have 6 lines. Build the appropriate object based on the tag and add it to the appropriate vector. Loop reading from the file until all contacts are read.

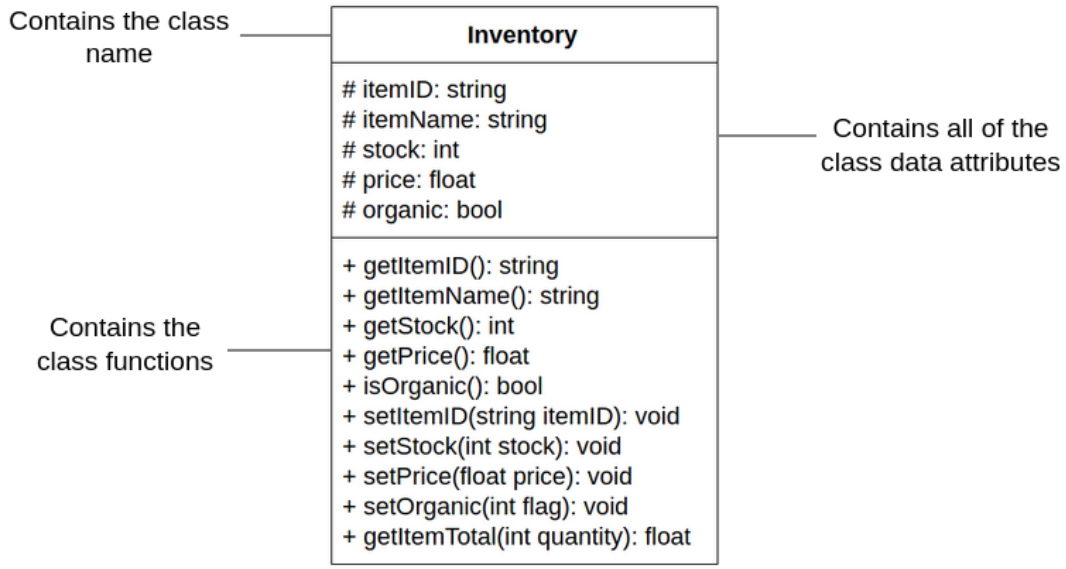
Once you're done reading from the file, close the file. Then, display all of the contacts to the user in a nicely formatted manner (example given in execution screenshot!). Make sure you distinguish between professional/personal contacts and count from one in your displays! Each should start at 1, independent of each other.

Example Execution:

```
Enter the file name: contacts
Invalid file. Please try again.

Enter the file name: contacts.txt

Personal Contacts:

Contact 1: Maggie Neal
Cell: 901-555-5555
Address:
108 A Road
Starkville, MS 39759
Birthday: November 5

Contact 2: Devin Neal
Cell: 601-234-0987
Address:
103 Some House
Memphis, TN 38135
Birthday: July 27

Contact 3: Ghost The Dog
Cell: 901-942-0149
Address:
5323 A Drum Road
Atoka, TN 38015
Birthday: Unknown


Business Contacts:

Contact 1: Kortni Neal
Work: 662-325-5555
Cell: 662-555-4534
Instructor at Mississippi State University

Contact 2: Shahram Rahimi
Work: 662-325-7777
Cell: 345-678-1290
Department Head at MSU, CSE
```

UML Cheat Sheet:

Each portion of a detailed UML Class diagram contains a different part of the class. There's a section for the name, variables, and functions belonging to the class. A breakdown of these is found below.



Additionally, each variable and function will be preceded by a symbol: +, -, or #. These refer to where your data will be accessible. Each of those mean the following:

- - Private
- + Public
- # Protected

For your variables, each will be followed by the data type they should be initialized as. So, "# itemID: string" should be a *protected string variable named itemID.*

Finally, your functions contain any/all parameters (and their data types!) following the function name. They then are following by a colon and the data type they should return. So, "+ getStock(): int" should be a *public function that has no parameters and returns an integer*.

Comment Block:

Your code should contain a comment block at the top containing information on who wrote the code, what the assignment is, when it is due, etc. Here is an example of a good comment block to put:

```
/*
 Name: <your name>                          NetID: <your netID>
 Date: <current date>                       Due Date: <enter in due date>

 Description: <What is the program?>
*/
```

Deliverables:

- C++ code
    - Your main file (.cpp file)
    - Contact, Personal, Professional classes (.h and .cpp files)
- A document (.pdf) with screenshots of the test main file and your built main file being run
    - The screenshots must be *legible* to count (too small or pixelated text will not be interpreted)
    - Show *all* possible error messages working

Point Breakdown:
(100 points total)
A submission that doesn't contain any code will receive a 0.

- 10pts - properly implemented inheritance
- 10pts - works with test file given
- 30pts - class specifications
    - 15pts - follows the UML for each class for functions (including naming, existence, etc)
        - 5pts PER class (Contact, Professional, Personal)
    - 5pts - all attributes are available and present
    - 10pts - Professional and Personal have two constructors
        - 5pts each
- 10pts - file handling
    - 5pts - file opening/closing (includes file verification loop)
    - 5pts - reads from the file correctly depending on format
- 20pts - vector handling
    - 5pts - properly builds 2 vectors of professional/personal type
    - 5pts - properly builds an object and then adds to the vector
    - 5pts - displays contacts to the user in a nicely formatted manner, similar to that shown in the example execution
    - 5pts - correctly calls object functions on vector items
- 10pts - execution screenshots
    - 5pts - test file
    - 5pts - built file (including showing error message)
- 10pts - programming style *

* Programming style includes good commenting, variable nomenclature, good whitespace, etc.