1325 Homework 4

HW Submission requirements:
1) Put your name and ID number on the top of EACH assignment (in comments for programs)
2) Name each file the name written in blue above each problem
3) Put all the files into a folder and zip it
3) Name the zipped folder with all assignments HW4.zip (-5 points for incorrect name)

NOTES:
1. **INCLUDE A README FOR EACH PROGRAM *(-10 points) Remember a README should let the user (my TA) know how to run your program***
2. **MAKE SURE YOUR CODE IS PROPERLY INDENTED. *(-10 points)***
3. **MAKE SURE ANY COMMENTS YOU INCLUDE ARE MEANINGFUL *(-10 points) Do not just put random comments on every line of code***
4. **DO NOT CODE EVERYTHING IN MAIN *(Automatic 0)***

NOTE: For the programs, part of the grading rubric will be putting functionality in the classes. For example, if you make classes but do not put functionality (or very little functionality), points will be deducted. Do not just make a class to make a class and then put all the work in main-the work should be contained in functions in the classes. Example: if you have a program where a person orders from a restaurant, you could have a Person class and a Restaurant class. The functionality of actually ordering can be a function in the Person class NOT something that happens in the main.

Problem 1 (18 points)-True/False Submit a document called Answers.doc

*Answer the following true/false questions. You must correctly state **WHY** your answer is true or false in order to receive credit.*

Code fragment 1:

```
class Fusilli: private Pasta
{
```

Code fragment 2:

```
class Boiling_pot{

 public:
 vector<std::shared_ptr<Fusilli>> boiling;
 Water *w;

 void add_noodles(int num_noodles)
 {
  for(int i=0;i<num_noodles;i++)
```

```
   {
     std::shared_ptr<Fusilli> ptr=std::make_shared<Fusilli>();
     ptr->num_order=(i+1);
     boiling.push_back(ptr);
   }
 }
```

Code fragment 3:

```
int main(int argc, char** argv)
{
  Boiling_pot b1;
  b1.add_noodles(3);

  cout<<b1.boiling[1]->num_order<<endl;
}
```

1. Assuming the class *Pasta* has a public variable called *p*, a *Fusilli* object would also have a public variable called *p*.
2. There is at least one indicator of polymorphism in this code.
3. If we know that *Pasta* has 3 member functions, we can be sure that all three are accessible in *Fusilli*.
4. *boiling* is a vector of *Fusilli* objects.
5. There is no way for us to give a value to the pointer *w* because there is no constructor or function that allows us to do so.
6. We can see that *+* is an overloaded operator.
7. The *ptr* variable holds a *Fusilli* object.
8. *num_order()* is an overloaded function since it was privately inherited.
9. *b1.add_noodles(3);* could be changed to *b1->add_noodles(3*);
10. *num_order* must be a public integer variable accessible in the *Fusilli* class.
11. Looking at the main function, we can say that the number 1 would be printed to screen.
12. If a class called *Farfalle* publicly inherited from *Pasta*, it would have the exact same number of member functions and variables as *Fusilli*.
13. *Boiling_pot b2=new Boiling_pot();* would be an acceptable line of code in main.
14. Assuming all the code for the program is kept in the same file, there must be at least 3 header files.
15. We can assume that *add_noodles()* is an overridden function.

Problem 2 (12 points)-Create classes. Submit a file named watches.cpp

ABC Watches sells watches. All watches are classified as designer, non-designer and antique. The difference between designer and non-designer is the price-anything with a price above $800 is classified as designer. Anything older than 50 years is classified as antique (you can choose if you want to strictly classify a watch as one thing or pick two-for example, a watch older than 50 years with a price over $800 dollars can be classified as both designer and antique OR either).

All watches have the ability to get repaired. When repairing a designer watch, you have to pay one fourth of the original price if it is made of gold and one third of the original price if it is made of silver. When repairing a non-designer watch, we check if the brand is on the "Free Repair List"-if it is not, we charge 15% of the original price. When repairing an antique watch, we charge half of the age (meaning if the watch is 60 years old, we charge $30).

Notes:
1) You should create a base watch class with characteristics and functions.
2) You should create any derived classes using this base class
3) Include the main you created to test out your classes for the TA

Notes:
0 POINTS IF YOU DO NOT: use inheritance and/or polymorphism as necessary

---
PICK ONE OF THE FOLLOWING (PROBLEM 3 OR PROBLEM 4): (70 POINTS) You are welcome to do both for extra practice (recommended), but only one will be graded.

Problem 3 Write a program. Submit a file named laundry.cpp

You just purchased an existing laundromat. Create a program for patrons of the establishment.

Notes:
1. Different types of patrons of this laundromat:
   a. Discount members (indicated with an id between D100-D500)
      i. They can only use the Slow and Very Slow machines
   b. Regular members (indicated with an id between D501-D900)
      i. They can only use the Slow and Fast Machines
   c. Elite members (indicated with an id between D901-D999)
      i. They can use any machine

2. Different types of machines of this laundromat:
   a. Fast (cost: 50 cents per use, needs 2 oz of detergent for each load)
   b. Slow (cost: 35 cents per use, needs 1.5 oz of detergent for each load)
   c. Very Slow (cost: 25 cents per use, needs 1 oz of detergent for each load)

Expected functionality:
1. All registered users should be read in from a file (command line argument). The file should look like:

```
Jon Doe,D160,3.75,5 //name, id, balance, amount of detergent in oz
Jane Doe,D450,5.00,3
```

2. The number of machines for each type indicated above should be read in from a file (command line argument). Include a sample file of how you chose to format the file.

3. The user of the program should be able to type in an ID. Based on the ID, the menu should show the types of machines available (for example, a discount member can only use the Slow and Very Slow machines-they should not see the Fast Machines) along with availability of each. If there is no matching ID (meaning the user was not in the file), this should be indicated.

4. A user of a machine should be able to change the status of the machine to *in use* (meaning they selected that machine for use). When a machine is not being used, it should be marked as *available*. Note that if a machine is *in use*, it should not be available for use to other users until the current user is done with it (see possible sample run). Once the user is done with a machine, it should be available for use once again.
    a. In order to use a machine, the user must have enough money and detergent. If they do not have enough, they should have the option to add more money or detergent.

5. A user should have the option to pick up laundry when finished (meaning the machine will become available once again). There should be an option for a receipt (in the form of a file). The receipt should include the name, machine used, current balance and detergent.

6. The program should exit when the user exits.

Notes:
You should make a Laundromat class, a Person class and a Machine class. (DO NOT ONLY USE THESE THREE CLASSES-0 POINTS IF YOU DO. Use inheritance and polymorphism as necessary.)

Possible sample run:

```
WELCOME!
Laundromat info read in: //should be a command line argument

***AVAILABLE***
Fast Machines: 3
Slow Machines: 0
Very Slow Machines: 2

All Registered Users info read in. //should be a command line argument

~Please enter your ID:
D150

~Welcome discount member.
```

```
  1) Start a laundry load
  2) Finish a laundry load //user finishes with a machine
start

***AVAILABLE***
Slow Machines: 0
Very Slow Machines: 2

~Which machine would you like to use?
Slow
~Sorry, we do not have any slow machines currently available.  Enter again.
Very Slow
~Ok, using 25 cents and putting 1 oz of detergent.

~Loading machine...
~You currently have: 75 cents left and 5 oz of detergent left.
-------------
~Please enter your ID:
D345

Sorry, we do not have this ID registered. //wasn't in the Registered Users file
-------------
~Please enter your ID:
D990

~Welcome elite member.
  1) Start a laundry load
  2) Finish a laundry load
start

***AVAILABLE***
Fast Machines: 3
Slow Machines: 0
Very Slow Machines: 1 //Notice it is one less because one is currently in use from the previous
member

~Which machine would you like to use?
fast
~Sorry, you do not have enough money on your account for this.  You need 50 cents and you
only have 20 cents.
~Would you like to add money to your account?
Yes
~How much?
50 cents
~Loading machine...
```

```
~You currently have: 20 cents left and 2 oz of detergent left.
-------------
~Please enter your ID:
D150

~Welcome discount member.
~You currently have one laundry load.  Would you like to pick up?
Yes //free up a machine
~Would you like a receipt?
Yes
~Printing out... //text file should be output
~Thank you for your business.
-------------
~Please enter your ID:
exit
```

Problem 4 Write a program. Submit a file named matching.cpp

ABC Language Company offers a service to match up people trying to learn a new language and a qualified language tutor.

Expected functionality:

1. The program should continuously read in files of people looking for language tutors (learners) until exit is somehow signaled (meaning the user of the program should be able to enter in different file names until exiting). A sample file of a person should look like the following (we will be testing your program with a file of this format):

   ```
   Paulie Glott  //Full name of the language learner
   Gender: Female //Desired gender of the tutor
   Language: Dinka //Target language
   Start: 08/6/2016  //Date learner started learning the language
   Max: $4.25 //Max amount willing to pay per hour
   Sunday: x    //x means available, - means not available
   Monday: -
   Tuesday: x
   Wednesday: -
   Thurday: x
   Friday: x
   Saturday: -
   ```

2. The program should allow language tutors to register. When the program exits, tutor information should be saved in a file, and when the program starts again, tutor information

should be populated in the program. This allows the tutor information to always be available (meaning that exiting the program does not mean that we no longer have a record of all people that signed up to be tutors). The information a tutor should give is:

   a. Name
   b. Gender
   c. Languages available to tutor (can be more than 1)
   d. **Date** he or she started tutoring
   e. Available days to teach
   f. Max number of learners at any given time
   g. Amount charged per hour

3. The program should attempt to match a learner (from the file read in) to one of the language tutors available in the program. If none are available, the learner is informed of this. The following criteria are used to match up learners and tutors:

   a. Match desired Gender
   b. Match target Language
   c. Check that time spent learning matches up with experience of the tutor. For example, if the learner started learning on 08/06/2016, they want a tutor with 2 years of experience (since we are now in 2018-it is up to you how to determine the current date)
   d. Match available days
   e. Check that the price per hour is acceptable for both parties
   f. Check that that the tutor is willing to take more learners. For example, if the tutor indicated they only wanted a max number of two learners, this should be considered

4. All possible matches should be output to screen-if none are available, it should be indicated.
5. The program can then ask which tutor to match with the learner (or not to match at all)
6. At any point in the program, a tutor can get rid of a learner (this would mean if they were tutoring 3 learners, they would then be tutoring 2)
7. The program should continue until exit

Notes:
You should make a Language_company class and Person class. (DO NOT ONLY USE THESE TWO CLASSES-0 POINTS IF YOU DO. Use inheritance and polymorphism as necessary.)

No possible sample run.