Practice Problems for PE07

For PE07, you'll have one problem that asks you to write a function to process vectors and structures. You have to know about how to pass the parameters correctly (input, output, input-output) and how to write functions that return a vector, using references and const correctly.

Below you'll find a few of the sample problems. Note, these are NOT necessarily the problems that appear on the test, but they should give you a feel for the structure and type of problems.

1 THE `findStars` PROBLEM

Using the `Star` structure defined in file `p07.cpp`, write the function named `findStars()`. The function takes two input parameters: a `vector` of `Stars` and a `string` representing a star name or part of a star name. The function returns a `vector` of `int` representing the `Draper` numbers of the stars where the name was found. Use `string::find()` to search the star names. Remember that `string::find()` return `string::npos` when the search parameter is not found.

```
vector<int> numbers;
numbers = findStars(vStars, "VEG");
```

2 THE `magnitudeBetween` PROBLEM

Using the `Star` structure defined in file `p07.cpp`, write the function named `magnitudeBetween()`. The function takes three input parameters: a `vector` of `Stars` as well as a `lower` and `upper` bounds. The function returns a `vector` of `size_t` containing indexes of the stars whose magnitude falls between the two bounds (exclusive).

```
vector<size_t> indexes;
indexes = magnitudeBetween(vStars, 1.5, 3.4);
```

3 THE `starsBetween` PROBLEM

Using the `Star` structure defined in file `p07.cpp`, write the function named `starsBetween()`. The function takes three input parameters: a `vector` of `Stars` as well as a `lower` and `upper` bounds. The function returns a `double` representing the average magnitude of the stars falling between the two bounds (exclusive). If no stars are in the range, then return `-1`. In addition, the function has an output parameter that returns the names of the stars that are included.

```
vector<string> names;
double avg = starsBetween(vStars, names, 2.3, 2.4);
```

Input parameter Output parameter

4 THE `longestHop` PROBLEM

Using the `Star` structure defined in file `p07.cpp`, write the function named `longestHop()`. The function takes one input parameter: a `vector` of `Stars` that represents a "travel itinerary". Visit every pair of stars in-order (0-1, 1-2, 2-3, etc.) and measure the distance between them. The function should return a `vector` of `Star` containing the two stars that are represent the longest "hop" or span between two stars in the trip.

We'll assume that the stars are in 2D space and that you measure the distance using this formula. You may write a function to do so.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
vector<Star> longest = longestHop(vStars);
```

5 THE `roundTrip` PROBLEM

Using the `Star` structure defined in file `p07.cpp`, write the function named `roundTrip()`. The function takes one input parameter: a `vector` of `Stars` that represents a "travel itinerary". Visit every pair of stars in-order (0-1, 1-2, 2-3, etc.) and add up the distance between them. When you reach the last star, include in your sum the distance between it and the first. The function returns the round-trip distance as a `double`. The function also has one output parameter that returns the average magnitude of all of the stars visited.

We'll assume that the stars are in 2D space and that you measure the distance using this formula. You may write a function to do so.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
double average;
double distance = roundTrip(vStars, average);
```

Input parameter Output parameter

6 THE `closestDistance` PROBLEM

Using the `Star` structure defined in file `p07.cpp`, write the function named `closestDistance()`. The function takes one input parameter: a `vector` of `Stars` that represents a "travel itinerary". Visit every pair of stars in-order (0-1, 1-2, 2-3, etc.) and measure the distance between them. The function should return a `vector` of `Star` containing the two stars that are closest to each other in the trip.

We'll assume that the stars are in 3D space and that you measure the distance using this formula. You may write a function to do so.

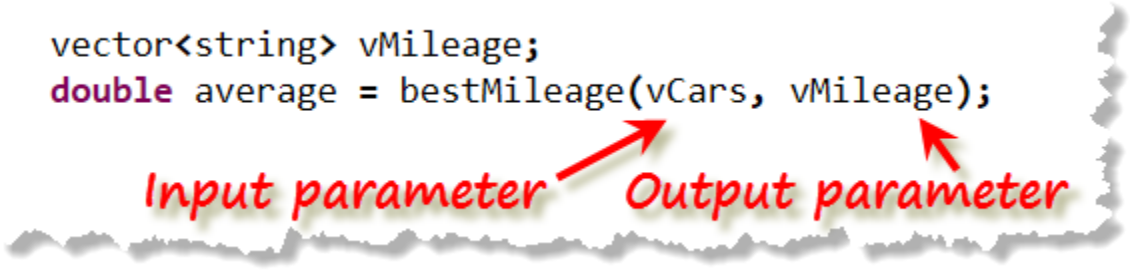
$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

```
vector<Star> closest = closestDistance(vStars);
```

7 THE bestMileage PROBLEM

Using the Car structure defined in file p07.cpp, taken from the 2017 EPA testing results, write the function named bestMileage(). The function takes one input parameter: a vector of Cars that represents the testing data. The vector is sorted by manufacturer, so all cars of a particular manufacturer are together in the vector. The function also has an output parameter, a vector of string that contains both the manufacturer and the model concatenated together, separated by a semicolon, like this: "BMW;430i Coupe"

Each entry is the car with the best mileage from each manufacturer. The function returns the average mileage of the selected cars.



Here is the Car structure.

```
// The structure (don't change these)
struct Car {
    string mfg, model;
    int horsepower, cylinders;
    double mpg;
};
```

8 THE bestValue PROBLEM

Using the Camera structure defined in file p07.cpp, write the function named bestValue().

```
// The structure (don't change these)
struct Camera {
    string manufacturer;
    string model;
    int releaseYear;
    int resolution;
    int weight;
    double price;
};
```

The function takes one input parameter: a vector of Camera. The vector is sorted by manufacturer, so all cameras of a particular manufacturer are together in the vector. The function returns a vector of string, with the manufacturer, model number and price concatenated together exactly in this format.

```
Agfa:ePhoto 1280:$179.00
```

(You'll need to use an ostringstream to do this.) We'll say that the best value is determined by dividing the resolution by the price. The higher number "wins".

```
vector<string> value = bestValue(vCameras);
```