Project #0 – Object Oriented Programming in C++

Computer Science 2413 – Data Structures

Fall 2015

This project is individual work. Each student must complete this assignment independently.

User Request:

"Create a simple system to read and write exoplanetary data."

Objectives:

1.	Use redirected standard C++ I/O (`cin`, `cout`) to read a file and produce output.	*5 points*
2.	Develop and use at least three classes to model different types of exoplanetary data. One of these classes should encapsulate the use of arrays.	*25 points*
3.	Use arrays of objects to store collections of the modeled data.	*10 points*
4.	Implement a program to manipulate exoplanetary data as described below.	*25 points*
►	Develop and use an appropriate design.	*15 points*
►	Use proper documentation and formatting.	*20 points*

Description:

Exoplanets are planets found beyond our solar system. For this project, you will put together several techniques and concepts you have learned in CS 2334 (or from a similar background) and some new techniques to make *ExoPlanIt*, an application that reads, stores, and output data on exoplanets. ExoPlanIt will read through a data file on exoplanets, derived from data found at http://exoplanets.org/, store the data from each entry in a structured way (e.g., all of the known exoplanets from the same system will be grouped together), and provide output derived from the input (e.g., besides listing the individual masses of the exoplanets in each system, ExoPlanIt will indicate the mean mass of the exoplanets in each system).

Operational Issues:

Your program will read the exoplanetary data file (a text file) via redirected standard input. The data file will be organized as follows:

The first line of the file will contain a comma-separated list of the headers for the data rows that follow. There will be 10 headers separated by 9 commas. Each header will be one or more alphanumeric characters and/or symbols but will not contain commas.

Each data row of the file will contain data on a single exoplanet. ExoPlanIt will not know in advance the number of data rows to read. However, for this project, you may assume that there will not be more than 500 data rows. The data in these rows will also be comma separated.

The first field in each data row will be the name of the star around which the exoplanet orbits, followed by a space, followed by the name of the exoplanet itself. Note that the name of the star may contain spaces but will not contain commas. Moreover, the name of the exoplanet will be a single character.

The second field of each data row will be an integer value that indicates the number of exoplanets known in the same system as the given exoplanet, followed by a comma. In the data file for this project, the data rows for exoplanets in the same system will be consecutive.

The third through ninth fields in each data row will be for floating point values that indicate various characteristics of the exoplanet in question. These fields will be for MSINI, A, PER, ECC, OM, T0, and K. (For the meanings of these abbreviations, see http://exoplanets.org/help/common/data.) Note that while these fields are for floating point values, some fields may contain integer values and some may be empty.

The tenth (and final) field in each row will be a binary (0 or 1) value that indicates whether there is a single star at the center of the given system (entry value of 0) or whether the system contains two more more stars (entry value of 1).

Consider the following data row:

```
61 Vir b,3,0.016075,0.05006,4.215,0.12,105,2453367.222,2.12,0
```

This data row indicates that the exoplanet in question is called “b” and orbits a single star called “61 Vir” along with two other known exoplanets. Its MSINI, A, PER, ECC, OM, T0, and K values are 0.016075, 0.05006, 4.215, 0.12, 105, 2453367.222, and 2.12, respectively.

As ExoPlantIt reads the data file, there are a couple of sanity checks that it should make on the data.

First, ExoPlanIt should ensure that there are 10 fields for each data row. As mentioned, some of these fields may be blank (fields 3 – 9) while the others are not allowed to be blank, and all fields should be present (that is, there should be nine commas before the final 0 or 1 of each row).

Second, ExoPlanIt should ensure that for each system, there is a match between the number of exoplanets listed in the second column of a data row and the number of data rows listing exoplanets for that system. For example, the data row shown above indicates that the system surrounding the star 61 Vir has three known exoplanets. This means that there should be a total of three data rows that have 61 Vir as the star name.

Third, ExoPlanIt should ensure that each planet’s name is one letter long and unique within that system. That is, there should be a single letter after the last space before the first comma of each row (the exoplanet’s name) and that single letter should not be repeated for any other exoplanet in the same system. (So, there could be a exoplanet named “b” in the system called “61 Vir” and another exoplanet named “b” in the system called “55 Cnc” but the other exoplanets in 61 Vir would not also be named “b.”)

If ExoPlanIt encounters any of these errors while reading the data file, ExoPlanIt should report which error it encountered and at which line number, then exit.

After reading in and storing all the exoplanetary data from the file, ExoPlanIt will print out all of the data it read in. However, the data it prints out will be slightly different in format from the data file format. In particular, before printing out the data for the exoplanets in each system, it will print a line of summary data on the system itself, and ExoPlanIt will leave two blank lines between the data for one system and the next. The summary data ExoPlanIt will print the following as a comma-separated list: The star name, the number of exoplanets known for that system, the average (arithmetic mean) MSINI of the exoplanets in the system, and the maximum and minimum PER of the exoplanets in the system. After the summary line for a system, ExoPlanIt will print the data for each exoplanet in the system, in the same comma-separated form as the input data file, with one exoplanet per line.

Implementation Issues:

For each class you implement, you must implement appropriate (and appropriately named) constructor, destructor, accessor, mutator, display, and, for those classes that contain arrays of data, size methods. The size methods will indicate how many items are stored in a given array. For example, if the data for the 61 Vir system is stored in the variable `exSystem`, then `exSystem.getExoplanets().size()` would return the value 3 because the example above shows that there are three exoplanets in the 61 Vir system.

Due Date:

You must submit an electronic copy of your source code through the dropbox in D2L by **Monday, September 28th by 2:45pm**.

Notes:

In this project, the only library you will use is iostream (`#include <iostream>`).

Be sure to use good object-oriented design in this project. That includes appropriate use of encapsulation, inheritance, overloading, overriding, accessibility modifiers, etc.

Be sure to use good code documentation. This includes header comments for all classes and methods, explanatory comments for each section of code, meaningful variable and method names, consistent indentation, etc.

You may write your program from scratch or may start from programs for which the source code is freely available on the web or through other sources (such as friends or student organizations). If you do not start from scratch, you must give a complete and accurate accounting of where all of your code came from and indicate which parts are original or changed, and which you got from which other source. Failure to give credit where credit is due is academic fraud and will be dealt with accordingly.