Programming Assignment #2

Program Description

This program analyzes the positional relationships between bodies in a 3-dimensional simulation. Each body is represented by a name and three coordinates (x, y, and z values). The program will open an input file (the filename provided as a command line argument) that contains a database of bodies. If the program successfully opens the file, it will parse the data into Body objects (a class you'll write), storing those objects in a list (an array). Using this list of bodies, the program will produce two reports that contain statistical information about the bodies (generated using the computational tasks described below), saved to two different output files (both filenames provided as additional command line arguments). The order the filenames are to be accepted from the command line follows:

1. body database input file
2. general statistics output file
3. sorted body listings output file

Computational Tasks

Your program must perform the following tasks, implemented by functions that you define:

1. Determine the two bodies, B_1 and B_2, which are closest to each other. Restriction: $B_1 \neq B_2$.
2. Determine the two bodies, B_1 and B_2, which are furthest from each other.
3. Determine the average distance between all of the bodies. For each distinct pair of bodies, B_1 and B_2, there is a single distance value used in this computation (if you've counted the distance between B_1 and B_2, you will omit the distance between B_2 and B_1.
4. Given a body B, produce a list of all known bodies, sorted by the distance each body is from body B (smallest to largest). B will be the first element in that list, since the distance from B to itself is zero.
5. [Extra Credit] Determine the volume of the smallest bounding box that contains all bodies.

Input

There is one input file for the program; the name provided on the command line. The program will need to open and read the data from the file. If the file does not exist or cannot be opened for any reason, the program should display an error message on the standard error stream and exit. The input file will have the format described below; your code does not need to check for incorrectly formatted data.

Body database input file

The body database input file contains entries for each body, one per line. Each entry will consist of 4 tokens, with the following format:

```
BODY_LABEL X_COORDINATE Y_COORDINATE Z_COORDINATE
```

- There may be any number of space or tab characters separating the tokens
- The body labels will not contain any whitespace
- The three coordinates will be floating-point values from -100000.0 to 100000.0 (units are kilometers)

A sample input file might look like the following:

```
Tau_Ceti          54553.4196888     -3098.93178117     -18716.7666998
Wolf_359          20977.8799308      69371.6784783     -31097.9321775
Proxima_Centauri   16696.185548      77363.2515435      -7601.8880558
Barnards_Star    -58011.3967387     -88836.1413007     -69029.5789315
```

There will be at least 3, but no more than 36, entries in the input file.

Output

The program will produce two output files; both names provided on the command line. If those files already exist, the program should overwrite their contents. If either file cannot be opened for any reason, the program should display an error message on the standard error stream and exit. You are expected to generate the output files with the exact formatting, as described below. Precision for all floating-point numerical output should be two decimal places.

General statistics output file

This output file will contain 3 or 4 lines, depending on whether you're doing the extra credit task. Line 1 should contain the labels of the two bodies that are closest together and the distance between them. Line 2 should contain the labels of the two bodies that are furthest apart and the distance between them. Line 3 should contain the average distance between the bodies. Line 4, if you're doing the extra credit, should contain the volume of the smallest bounding box that contains all bodies:

```
CLOSEST_BODY_1_LABEL CLOSEST_BODY_2_LABEL DISTANCE_BETWEEN_THEM
FURTHEST_BODY_1_LABEL FURTHEST_BODY_2_LABEL DISTANCE_BETWEEN_THEM
AVERAGE_DISTANCE
VOLUME_OF_BOUNDING_BOX
```

- All numerical values should indicate their respective units of measurement
- There should be no empty lines in the output
- The three tokens on the first two lines should be separated by exactly 1 space character.

A sample of this output file might look like the following:

```
Barnards_Star Proxima_Centauri 192293.58km
Proxima_Centauri Wolf_359 25184.56km
119824.62km
1149201781111127.50km^3
```

Sorted body listings output file

This output file will be written in sections, with one section for every body from the input file, alphabetically ordered. Each section will contain the label for the body (origin), followed by a list of all bodies, ordered by the distance between them and the origin body (one per line).

A section will have the following format, with <T> indicating a tab character:

```
LABEL_OF_ORIGIN_BODY
<T>LABEL_OF_BODY_1<T>DISTANCE_BETWEEN_BODY_1_AND_ORIGIN_BODY
<T>LABEL_OF_BODY_2<T>DISTANCE_BETWEEN_BODY_2_AND_ORIGIN_BODY
...
<T>LABEL_OF_BODY_N<T>DISTANCE_BETWEEN_BODY_N_AND_ORIGIN_BODY
```

- N is the number of bodies found in the input file
- A tab character should proceed each line after the origin body label
- A tab character should separate the labels and the distances
- The units should not be indicated for distance values

A single empty line should separate two sections.

A sample of this output file might look like the following, with <T> indicating a tab character:

```
Barnards_Star
<T>Barnards_Star<T>0.00
<T>Tau_Ceti<T>150176.85
<T>Wolf_359<T>180853.06
<T>Proxima_Centauri<T>192293.58

Proxima_Centauri
<T>Proxima_Centauri<T>0.00
<T>Wolf_359<T>25184.56
<T>Tau_Ceti<T>89615.14
<T>Barnards_Star<T>192293.58

Tau_Ceti
<T>Tau_Ceti<T>0.00
<T>Wolf_359<T>80824.50
<T>Proxima_Centauri<T>89615.14
<T>Barnards_Star<T>150176.85

Wolf_359
<T>Wolf_359<T>0.00
<T>Proxima_Centauri<T>25184.56
<T>Tau_Ceti<T>80824.50
<T>Barnards_Star<T>180853.06
```

User Interface

There is no user interface for this program. The only interaction is specifying the input and output filenames as arguments on the command line.

Grading

The program will be graded based on the following rubric:

- Does the program compile (on one of the cslab machines): 15%
- Correctness and implementation of the class Body: 10%
- Correctness and implementation of the sorting and searching routines: 15%
- Correctness and implementation of the file I/O operations: 15%
- Correctness and implementation of the main function and any other functions in the program: 20%
- Correctness and use of multiple file implementation: 10%
- Documentation: 10%
- Style (variable naming, indentation, etc…): 5%
- [Extra Credit] Correctness and implementation of the bounding box task: 10%

Programming Rules

Your program must adhere to the programming rules as described in the "Programming Rules and Guidelines" document posted on Blackboard. Of note, or in addition:

Your program must be well documented. This includes having a preamble, prologue for every function prototype (describing parameters, pre- and post-conditions, and return value), and comments throughout the code (describing any non-trivial algorithms in plain English).

Your program must follow common stylistic guidelines that cover line spacing, whitespace, indentation, and choosing appropriate variable/function/class names using consistent naming conventions. You may not use global variables, unless they are constants.

You must implement and use a class, named **`Body`**, to represent a *single* body. You'll need to determine the member variables and member functions your class will need and which members will be made public versus private. As you make those decisions, remember that your class must respect the OOP principles for access, information hiding, data abstraction and encapsulation.

You must store the bodies (objects of your class **`Body`**) in an array in your program. You may assume there will be at most 36 bodies in the input file. You may not use Vectors anywhere in your program.

You must implement and use your own sorting and searching functions.

All input and output stream objects must be created in your main() function. Opening, closing and error handling for all input and output files must be performed in your main() function, as well. All other I/O operations *should* be performed in your main() function (or functions specifically designed for it, that do not perform any other tasks).

The program must read the names of input and output files from the command line (you cannot hard-code the names of the files into the program). You are expected to handle a missing input file gracefully, as described in the Input section above. Any problems encountered attempting to open the output files should also be handled gracefully, as described in the Output section above. Otherwise, you may assume the input file is correctly formatted.

All output should be sent to a corresponding file output stream, except for error messages. Any error messages you display should be sent to the standard error stream. A successful execution of your program should have no console output.

You must use multiple source code files for your program (separate compilation). The class and function declarations should be placed in a separate header file. The class and function implementations should be placed in a separate implementation file. The source code file containing your main function must be called **`main.cpp`**.

What to submit and when

This assignment is due by the end of the day (i.e. 11:59PM, EST) on November 11th, 2014.

Before you submit the assignment, make sure that it compiles and runs correctly on one of the cslab machines. Do not enhance your program beyond this specification. Do not make it do anything except what is described in this document.

You need to submit multiple files for this assignment, so they must be placed into a directory and zipped up as instructed below. The name of this directory must be **`lastname_firstname_hwk2`** (replace **`lastname`** and **`firstname`** with your last and first name, *using lowercase letters*). Do not name it anything else. If it is named anything else, you lose 3% of the program's value.

You are to create the above named directory and place all of your source code files into it. The source code file that contains your main function must be called **`main.cpp`** (you will lose 1% if it is not). Do not place any executable files, object files, input or output files in this directory. You will lose 1% for each file that does not belong there. With all necessary files in your directory, if you're on a Linux or Mac computer, run the following command from the parent directory that contains your newly created directory:

```
zip -r lastname_firstname_hwk2.zip lastname_firstname_hwk2/
```

This will compress all of your files into a file named **`lastname_firstname_hwk2.zip`**. If you're on a Windows computer, there should be an option to "compress" the folder to a zip archive.

Submit the zip file to the assignment in our Blackboard course. Do not paste the code into your submission.