2E3 Project

2E3 term project is an exercise in the use of C++ object oriented programming and data structures.

Mobile crowdsensing (MCS) applications aim at coordinating and activating the participation of communities of volunteers willing to use their smart-phones to harvest data as they move in urban areas. The objective of this assignment is to implement a simulation of a MCS system that may be used by urban cyclists to assess their riding experience and to identify potholes on the road on which they cycle. Rather than an actual mobile application, this assignment requires you to implement a simulation, i.e. a purely software implementation that should exhibit a similar behaviour to that of a real system, but without relying on actual mobile or sensing devices.

To this end your task will be to implement the following classes:

GPS: This class will simulate the GPS unit installed on the users' smart-phone. To this end, the class needs to implement a *getLocation*() method that will return the latitude and longitude where the cyclist is currently located (e.g. assume that *getLocation* will be called every second). For instance, the first time *getLocation*() is invoked, it could return (53.344384, -6.261056), the second (53.344424, -6.260818), the third (53.344450, -6.260614), the fourth (53.344476, -6.260324), the fifth (53.344501, -6.260088), the sixth (53.344537, -6.259906), and so on (these example locations correspond to coordinates along College Green simulating a cyclist approaching the entrance of Trinity College Dublin).
You should read the locations from a file.

[20% of your mark]

Sensor: This class will simulate a sensing device equipped with a 3-axis accelerometer. The class Sensor needs to implement a *readAccelleration*() method to return the X, Y and Z components of the acceleration vector measured by the sensor (e.g. assume that *getAccelleration* will also be called every second). For instance, the first time *readAccelleration*() is invoked, it could return (0, 7, 11), the second (1, 7, 10), the third (2, 6, 40), the fourth (1, 7, -33), the fifth (0, 7, 12), the sixth (1, 8, 10) and so on (note how the Z component in the third and fourth values in this example shows the presence of a spike, which could signify the event of riding over a pothole).
You should read the accelerometer readings from a file.

[20% of your mark]

Note: You should make sure and assume here that the files you use to store the locations and the sensor readings will contain the same number of items.

In addition, your program should be able to work with a variable number of entries, i.e. it will work correctly if an equal number of additional entries are added to both the location and the sensor data.

Main: The main of your program will need to simulate the MCS app. Your implementation will need to create and initalise a GPS object and a Sensor object and use them in order to:

1. Query (every second) and print on the screen the users' location and the accelleration vector measured, respectively, by the GPS and by the accelerometer sensor.

[10% of your mark]

2. When you have processed all the available input, print the average Z-axis acceleration measured by the sensor along the entire simulated ride.

[10% of your mark]

3. Print all the coordinates and the corresponding acceleration values (only for the Z-axis) where the absolute value (of the Z-axis component) of the accelerometer was "measured" to be greater than a given threshold T (where T is either a per-defined constant or a variable whose value you could ask the user).
The list of coordinate/acceleration should be printed in descending order of acceleration (from the higher value to the lowest value)

[20% of your mark]

For instance, in the example used in the description of the GPS and Sensor classes, your program will have to print the following:

```
Z-axis acceleration threshold? 30

(53.344384, -6.261056): 11
(53.344424, -6.260818): 10
(53.344450, -6.260614): 40
(53.344476, -6.260324): -33
(53.344501, -6.260088): 12
(53.344537, -6.259906): 10

Average Z-axis acceleration is:8.33

Only the value where the absolute value of Z-axis component of the acceleration is greater than 30, in descending order:
(53.344450, -6.260614): 40
(53.344476, -6.260324): -33
```

Additional features: Any additional feature you may like to implement to showcase your ability to define and use C++ classes and data structures.
Please make sure that those features are clearly demonstrated when your code is executed.

[Additional features will count 10% of your mark]

This project accounts for 10% of your overall 2e3 mark.

You will get full marks for your code for properly implemented classes and features (e.g. class defined in .h/.hpp file, class implemented in a .cpp file, correct use of data structures, no logic errors …).

You are free to decide how best to implement each class, and any other class you may need to complete this exercise. However, you need to implement all the data structures (e.g. linked lists) you use, i.e. you are not allowed to use the collection classes from the standard C++ library.

IMPORTANT POINTS:

1. The code that does not compile or show implementation of any of the features will get 0 points.
2. The project needs to be submitted online to Blackboard, under the folder Labs/Project.
3. Deadline for submission is 23:59:59 on January 31st 2013.
4. You need to submit:
   - all your source code (.h/.hpp/.cpp files)
   - the executable of your program, ready to be executed on the operating system (e.g. Mac OS, Windows, Linux) you used on the computer where you built your program (see Labs/Week3/Lab2)
   - a readme.txt file to briefly (max 1 page) describe (i) the way you have solved this problem (including what extra classes and assumptions you needed to introduce in your solution, (ii) explain why your choice of data structures and your design are appropriate in this context, (iii) any additional feature you have implemented, and (iv) in which OS (Mac/Windows/Linux) your executable must be executed.

   [This readme.txt will count for 10% of your mark]
5. I strongly recommend that you finish and submit the assignment in plenty of time, ideally before the start of 2nd semester so it does not interfere with coursework you might be given in semester 2. No allowances will be made or extensions given if your internet connection is down on the evening of January 31st etc as you have full 7 weeks to work on/submit the project.
6. I suggest you to build your code incrementally. For instance, you could start by implementing simpler versions of your GPS and Sensor classes, which will not read the data from a file but return simple random data. You could use these classes to implement and test your main program, before you implement the final version of your GPS and Sensor classes.
7. Late submissions will get 10% penalty per day.
8. Incomplete submissions will get 0 points, or will be treated as late submissions and penalized 10% per day until the full project is submitted! Therefore make sure to submit ALL .h/.hpp and .cpp files on time.
9. Make sure to submit CORRECT VERSIONS of ALL your .h/hpp and .cpp files if you worked with several different versions. Files coming from different versions

might not compile when put together, and CODE THAT DOES NOT COMPILE WILL RECEIVE 0.

10. While working on your project backup your code frequently so you don't need to start from scratch if your laptop breaks, you lose your usb key etc.
11. Projects will be run through plagiarism detector so please make sure to submit only your own individual work!