School of
Engineering

Newcastle University

EEE2007 – COMPUTER SYSTEMS AND MICROPROCESSORS

PROJECT 2: USING C++ TO ANALYSE DATA

Dr Rishad Shafik

Total Marks: 100

Project Objectives

1. To develop a working knowledge of structured C++ programming with object-oriented features, and
2. To gain experience with simple data analysis application through C++.

Many Internet of Things (IoT) companies are emerging in the UK and around the world that use commercially-off-the-shelf sensing systems to generate user-based electronic solutions. You are given a set of files in the attached archive (`Sensor.hpp`, `Sensor.cpp`, `main.cpp` and `Makefile`), which simulate the sensor behaviour for a dummy IoT device. Note the following:

A. First spend some time reading through the codes (`Sensor.hpp`, `Sensor.cpp` and `main.cpp`). You should be able to run the code by simply using "`make`", followed by "`./program`". The already worked out part of the codes simulates the behaviour of a sensor system attached to a human subject. The sensed data are directly stored in a file with each line meaning the following

```
Time_tamp        sensed value (with required precision)        sampling time
```

For example, in the line:

```
2017-11-27.19:17:32        39            100
```

i. The first time stamp is basically the current date and time to seconds
ii. The second number gives the sensed data
iii. The third number gives the sampling time for this observation (100ms) [this information is only relevant as the sample values are affected; lower sample time can generate more correlated values; higher sampling time will generate more uncorrelated values]

As you run more and more, the data keeps appending to a .dat file (in this example `sensorA.dat` file). So if you want to clear up the data you can just enter "`make cleandata`". You can also recompile by cleaning old object files by "`make clean`", followed by "`make`".

B. Make the main function capable or accepting arguments from the user as below:
`./program <sampleTime> <PrecisionBits> <numberOfSimulatedSamples>`
See your main.cpp for more details in comments.

[15]

C. Create a *Person* class, with the following attributes:
   a. Private variables: name and age;
   b. Public methods: `getAge(..), setAge(..) showPersonInfo(..).`
   c. Key public methods: `int getTimeWindow(date)` and `int analyseSensedData(time_window, sample_value)`

D. The `int getTimeWindow(date)` function should determine the time window with current time stamp (see `sensorA.dat` for example); the `analyseSensedData(..)` function should read each sample in the data file (in this case `sensorA.dat` file) and check for the following criticality conditions:

Time window	*Age Range*	*Non-critical range*
(Night) Midnight – 7am	0 – 6	20 – 30.2
	7 – 16	22 – 35.9
	17 - 38	22.7 – 37.3
	39 – 55	25.7 – 38.3
	55+	20 – 35
(Morning) 7am+ - 10am	0 – 6	30 – 52
	7 – 16	30 – 55
	17 - 55	32 – 57.8
	55+	30 – 50
(Daytime) 10am+ - 6pm	0 – 6	20 – 40
	7 – 16	22 – 39
	17 - 55	22.7 – 41
	55+	25.7 – 37
(Evening) 6pm+ - midnight	0 – 6	25 – 52
	7 – 16	25 – 55
	17 - 55	25 – 57.8
	55+	25 – 40

[30 marks]

E. Following step B, parameterise the number of persons to test, followed by their ages as follows:

```
./program <sampleTime> <PrecisionBits> <numberOfSimulatedSamples> <NumberOfPpersons>
<Age1> <Age2> … so on
```

You must make sure that the user inputs are validated. For example, if there number of persons is 5, there should be 5 numbers following for age inputs.

[15 marks]

F. Your program should test for five persons of different age groups shown above (but you can test with more), and write the following output in `analysis.txt` file:

```
---------------------------------------------------------------------------------------
Person 1: (0-6 years)
---------------------------------------------------------------------------------------
Night                   15% critical samples
Morning                 24% critical samples
Daytime                 33% critical samples
Evening                 17% critical samples
---------------------------------------------------------------------------------------
Person 2: (7-16 years)
---------------------------------------------------------------------------------------
Night                   32% critical samples
Morning                 31% critical samples
Daytime                 11% critical samples
Evening                 37% critical samples
---------------------------------------------------------------------------------------
Person 3: (17-38 years)
---------------------------------------------------------------------------------------
Night                   11% critical samples
Morning                 42% critical samples
Daytime                 31% critical samples
Evening                 31% critical samples
```

```
-------------------------------------------------------------------------------------
Person 4: (39-55 years)
-------------------------------------------------------------------------------------
Night                   9% critical samples
Morning                 31% critical samples
Daytime                 16% critical samples
Evening                 19% critical samples
-------------------------------------------------------------------------------------
Person 5: (55+ years)
-------------------------------------------------------------------------------------
Night                   18% critical samples
Morning                 22% critical samples
Daytime                 19% critical samples
Evening                 55% critical samples
-------------------------------------------------------------------------------------
```

[Calculation/Coding: 20 and Display: 10 Marks]

Note that the calculations above are used for examples only. Your presentation will need to be similar within the file (however the numbers are likely to change).

F. Your new class should be named as `Person.hpp` and `Person.cpp` files and these should be incorporated in your Makefile. Your Makefile should also be able to clean up the output `analysis.txt` file when cleaned.

[10 marks]

Deliverables

You should submit a single archive with `Makefile` and the rest of the source code files consisting of the correct program. No report is necessary. To avoid errors, make sure your original files are first saved in a safe place, and then you can create the archive using copies of these files.

Marking

This assignment will constitute 10% of the total mark for the module. The marks' distribution is already shown in the problem descriptions, totalling 100. The actual mark will be scaled to the module's marks afterwards.

MARKING INSTRUCTIONS

Correct, and properly commented code with correct presentation will merit up to 90% marks. Well-organised and commented code will merit another 10% marks.

Plagiarism is strictly prohibited as it may result in **serious penalties**. There would be informal tutorial discussions on this assignment after the regular lecture hours. The submission deadline is strictly **15 Dec 2017**, after which the submissions would be considered late and usual late submission rules would apply.

Feedback

After your assignment has been marked, feedback is provided as follows:
1) *Blackboard*: the document submitted has been annotated with little 'blue balloons' in the usual manner. I will notify you when this information becomes available.
2) *Email*: Shortly after the Blackboard feedback becomes available, a feedback sheet (a single page pdf file) with a detailed breakdown of your marks will be communicated to everyone.