Mini project

Speed sensor simulation and data analaysis

I. Description

Design and write program(s) in C or C++ with proper functions and data structure to simulate a speed sensor mounted on a DC motor with the following specification:

- Measurement range: 0 ÷ 3000 rpm (revolution per minutes)
- Resolution: 0.2 rpm

The required tasks as below:

1. Task 1:

Write a program which allows user to provide the number of sensors (corresponding to number of DC motors which need to be monitored), the sampling time and measurement duration (simulation interval) and the starting time. These inputs should be provided by using command-line as below:

C:\\speed_sim –n [num_sensors] –st [sampling] –si [interval]

- `speed_sim`: is the execution file name without extension.
- `-n [num_sensors]` are two arguments to set the number of sensors and must be provided together, `[num_sensors]` should be replaced by a specific number. An error message should be shown if only one of these two are presented. If both of these two are missing, the program will use the default value for the number of sensors which is 1.
- `-st [sampling]` are two arguments to set the sampling time and must be provided together, `[sampling]` should be replaced by a specific number. An error message should be shown if only one of these two are presented. If both of these two are missing, the program will use the default value for the sampling time which is 10 seconds.
- `-si [interval]` are two arguments to set the simulation interval and must be provided together, `[interval]` should be replaced by a specific number. An error message should be shown if only one of these two are presented. If both of these two are missing, the program will use the default value for the simulation interval which is 1 hour.

The program will generate a data set consisting of sensor id, the timestamp and random values of speed with the starting time of the simulation is the current time which is based on system time.

- The sensor ids are the number from 1 to the number of sensors, i.e. if the number of sensors provided by the user is 10, we have 10 sensors with the ids are 1, 2, 3, …, 10. The data set is saved to a file name “speed_data.csv”.
- The timestamp is in format of YYYY:MM:DD hh:mm:ss, where:
  - YYYY – year, MM – month, DD – day.
  - hh – hour, mm – minute, ss – second.
    E.g: 2024:04:15 08:30:00
- This file should follow the format of a comma-separated values (CSV) file, please refer here for more information: https://www.ietf.org/rfc/rfc4180.txt

For example: C:\\speed_sim –n 3 –st 1 –si 30

The above command will generate “speed_data.csv” with the following content:

```
id,time,value
1,2024:04:15 08:30:00,1500.6
2,2024:04:15 08:30:00,200.2
3,2024:04:15 08:30:00,1200.2
1,2024:04:15 08:30:03,100.2
2,2024:04:15 08:30:03,1600.2
3,2024:04:15 08:30:03,1200.4
…
```

2. Task 2:

Write a program to process the data in “speed_data.csv”. The program should be run by using command-line as below:

`C:\\speed_process [data_filename.csv]`

- `speed_process`: is the execution file name without extension.
- `[data_filename.csv]` is the csv file which consists of the speed data set. If the user does not provide the filename, the program will use the default name which is “speed_data.csv”.

For example: `C:\\speed_sim speed_data.csv`

The program should be able to handle at least 10000 data points which may be included in the “speed_data.csv” file. This program should implement the following sub-tasks:

a. Task 2.1:

Assuming that the speed of the motor in normal operation are varied within $850 \div 1650$ rpm (lower bound and upper bound are included). The program needs to validate the speed data in the data file, data points with the speed values which are not in the valid range have to be filtered as outliers. The outlier data points are stored in a file named “outlier_data.csv” which should follow the below format:

```
number of outliers: 4
id, time, value
1,2024:04:15 08:30:00,1700.6
3,2024:04:15 08:30:00,200.6
3,2024:04:15 08:30:03,1900.4
2,2024:04:15 08:30:03,100.2
```

The first line is “number of outliers: X” where X is the number of invalid data points, e.g. 4 in the above example.

All the valid data should be put in another csv file named “valid_speed_data.csv” with the same format as the input data file, i.e.: same format as the one generated in task 1.

Note: all the data points used from task 2.2 onwards are valid data only, the outliers identified in task 2.1 should NOT be included.

b. Task 2.2:

Identify the largest, smallest, and average speed for each sensor id (apply to the valid data points only) every hour for the whole simulation duration, and store the results in a file named “data_summary.csv”. The example is as the following:

The time of the max and min values in the file above are the earliest timestamps those values appear in an hour. Meanwhile the time of the mean is the starting time of that one-hour interval.

```
id,parameter, time, value
1,max,2024:04:15 08:30:00,1500.6
1,min,2024:04:15 08:51:03,950.8
1,mean,2024:04:15 08:00:00 , 1200.5
1,max,2024:04:15 09:10:00,1400.6
1,min,2024:04:15 09:51:03,850.8
1,mean,2024:04:15 09:00:00 , 1109.5
2,max,2024:04:15 08:35:00,1590.6
2,min,2024:04:15 08:32:03,900.8
2,mean,2024:04:15 08:00:00, 1100.5
2,max,2024:04:15 09:25:00,1490.6
2,min,2024:04:15 09:32:03,890.8
2,mean,2024:04:15 09:00:00, 1202.5
…
```

c. Task 2.3:

We assume that the motor can change (increase or decrease) the speed by maximum 100 rpm per second. Please count the number of times that speed increment and decrement between two

consecutive data points exceed 100 rpm per second for each speed sensor (motor). The alert value should be specified with respect to the frequency values of either decrement and increment as below:

Table 1. Frequency of abnormal and respected alert

Frequency (either increment or decrement)	Alert	Alert code
< 5	normal	0
$5 \leq frequency < 15$	warning	1
$15 \leq frequency < 30$	bad	2
$30 \leq frequency < 50$	severe	3
$50 \leq frequency$	failure	4

The results should be stored in a csv file named "data_statistics.csv" with the following format, the sensor 1 for example should have alert values of "warning" in both lines of "increment" and "decrement" direction as the frequency value of abnormal increment is 6

```
id,direction,frequency,alert
1,increment,6,warning
1,decrement,0,warning
2,increment,3,bad
2,decrement,21,bad
3,increment,0,normal
3,decrement,4,normal
…
```

d. Task 2.4 (optional):

Choose a sorting algorithm and sort the valid data in the data_file.csv if the argument –s is added as below.

```
C:\\speed_process [data_filename.csv] -s
```

The sorted data should be stored in a file name "sorted_speed_data.csv". The data should be sorted ascendingly by id and values, i.e. the smallest sensor id should appear first, and all the values of the same sensor id should be sorted from the smallest to the largest. The time taken to run the algorithm should be measured and write into "task2.log" as "Sorting duration: X seconds" where X is a specific number of the time measured. "task2.log" is the log file for task 2 as mentioned in Section II.

NOTE: All the sub-tasks in task 2 should be integrated in a single program, do NOT implement each sub-task in a separate program.

3. Task 3:

It is assumed that the users need to send and receive the **valid** speed data obtained in ***Task 2.1*** over a communication protocol. The data packet transferred is a byte array which must follow the below structure:

Table 2. Data packet frame

Start byte	Packet Length	ID	Time	Speed	Acceleration	Checksum	Stop byte
0x0A (1 byte)	1 byte	1 byte	4 bytes	4 byte	4 bytes	1 byte	0x0F (1 byte)

Where:

- Start byte (1 byte) is the first byte in the packet and always has the value of 0x0A.
- Stop byte (1 byte) is the last byte in the packet and always has the value of 0x0F.
- Packet length is the size of the packet including the start byte and stop byte.
- Id is the identification number of the sensor (sensor ID) and must be a positive value (>0)
- Time is the measurement timestamp in second which follow Unix timestamp format, by default it should be local time (Vietnam time or GMT + 7).

- Speed is the speed value which is a 4-byte real number represented with IEEE 754 single precision floating-point standard.
- Acceleration is rate of change of the speed compared to the previous values at the previous timestamp, i.e.: it can be calculated as $\frac{current\ speed - previous\ speed}{current\ time - previou\ \ time}$.
    - If the data point is the first one in the data file, the acceleration is 0.
    - The acceleration calculation should be round up to have only 1 decimal point, e.g.: assuming that $current\ time - previous\ time = 3\ [second]$, current speed is 1600.2 and the previous speed is 1590.1, the acceleration should be:
$$\frac{1600.2 - 1590.1}{3} \approx 3.4$$
- Checksum is the byte to verify the data packet and is calculated by using two complement algorithm of the byte group including [packet length, id, time, Speed, Acceleration]

All the numbers (integer and real ones) are represented as big-endian.

The user executes task 3 with the following command-line statement:

```
C:\\ speed_comm [input_file] [output_file]
```

Where

- `[input_file]` is the name of the input file.
- `[output_file]` is the name of the output file.

For example:

```
C:\\speed_comm speed_data.csv hex_packet_ee3490e.dat
```

The programm should:

- Read each line of the input file, i.e.: speed_data.csv
- Convert each data line into the data packet as described in Table 2, each byte is separated by a space character and represented as hex number, e.g.: *assuming that a line of "**2,2024:04:15 08:30:03,1600.2**" need to be converted to a data packet, this line is the kth line in the file named "speed_data.csv", the previous line of sensor id 2 is "**2,2024:04:15 08:30:00,1590.1**"*

0A 11 02 66 1C 83 1B 44 C8 06 66 40 59 99 9A 0F

- Write each packet in one line in the output file, i.e. hex_packet_ee3490e.dat
- Override the output file hex_packet_ee3490e.dat if it has been existed.
- Be able to process at least 10000 data points which means that the input file speed_data.csv may consist of at least 10000 data lines with valid data.

II. Additional requirements:

The program needs to store any run-time errors occurring in log files (normal text file format). There must be 3 different log files for 3 tasks in section I, they are named as task1.log, task2.log and task3.log for task 1, task 2 and task 3 respectively. Each error must be written in a line in the corresponding log file with the format below:

Error AB: DESCRIPTION

Where:

- AB is the error code which is a 2-digit number (if the number is smaller than 10, there must be a leading zero digit, i.e.: 01, 02, …)
- DESCRIPTION is the detail of the error.

1. **Errors may happen when executing task 1:**
    - Wrong command-line statement, e.g.: lack of the 1 or a few required command-line argument. The error message can be "Error 01: invalid command".
    - Invalid value of the command-line argument, e.g. negative number of sensors. The error message can be "Error 02: invalid argument".
    - "speed_sensor.csv" file is existing and is a read-only file. The error message can be "Error 03: file access denied"
2. **Errors may happen in task 2:**
    - The input file data_filename.csv does not exist or is not allowed to access. The error message can be "Error 01: input file not found or not accessible"

- The input file data_filename.csv or location.csv does not have proper format as required, e.g. it has no header file, wrong number of comma, or even consists of completely different data format. The error message can be "Error 02: invalid input file format"
- The user types the wrong command-line format, e.g.: the filename is missing. "Error 03: invalid command"
- The input data file contains wrong data:
    - All the data fields in one line are blank, e.g.: ", , "
    - Id is blank or invalid, e.g.: "-1,2024:04:15 01:00:00,950.01"
    - Time is blank or invalid, e.g.: "1,2024:04:15 01:00:,950.01"
    - speed value is blank or invalid, e.g.: "1,01:00:00, "

The error message must include the line number in the input file in which the error happens, i.e.: "Error 04: invalid data at line X" where X is the line number. The first line in the input file which is the header line is line 0 (zero), e.g.: in data_filename.csv the line "id,time,value" is the header line, the next lines onwards are data line and are numbered from 1 (one).

The program should then ignore the line with wrong data and continue to process next line when perform task 2.

3. Errors may happen in task 3:

- The input file does not exist or is not allowed to access. The error message can be "Error 01: input file not found or not accessible"
- The input file does not have proper format as required, e.g. it has no header file, wrong number of comma (if it is a csv file), or even consists of completely different data format. The error message can be "Error 02: invalid input file format"
- The user types the wrong command-line format, e.g.: one or both the two filenames are missing. "Error 03: invalid command"
- The output file is existing and cannot be overridden. Error message can be: "Error 06: cannot override output file"
- The input file is a csv file and it contains wrong data:
    - All the data fields in one line are blank, e.g.: ", , , "
    - Id is blank or invalid, e.g. "-1, 2023:11:11 00:00:00,1, 50.10"
    - Time is blank or invalid, e.g. "1,2023:11:11 00:00:,2, 50.10"
    - Speed value is blank or invalid, e.g. "1,2023:11:11 00:00:00,12, "

The error message must include the line number in the input file in which the error happens, i.e.: "Error 04: invalid data at line X" where X is the line number. The first line in the input file which is the header line "id,time,value" is line 0 (zero), the next lines onwards are data line and are numbered from 1 (one).

The program should then ignore the line with wrong data and continue to process next line.

- If the input file contains duplicated data, i.e. two or more lines in the file are exactly the same, the error message must include the line numbers in the input file in which the error happens, i.e.: "Error 05: data at line X and Y are duplicated" where X is the line number where the data first appears and Y is the line number where data is the same as X.

  The program should then ignore the line with duplicated data and continue to process next line.

4. Other errors:

The students can suggest more errors which may happen but not be listed above. Those errors should be stored in the respective log file and described in the report.

III. Program design:

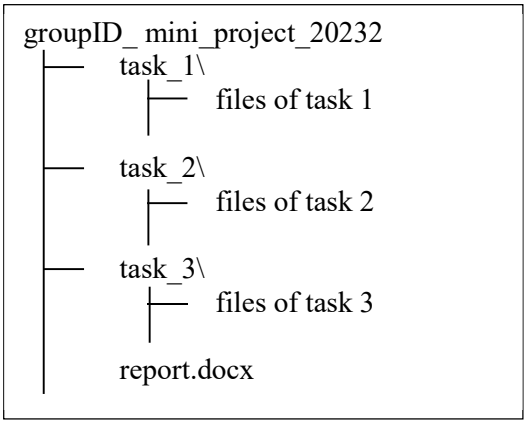
The students must use top-down approach to design the program.

It is required to draw the top-down diagrams to illustrate the relationship between the functions in the program for each task together with brief description in the report.

The students need to draw at least one flowchart in each tasks:

- One flowchart of the overall program of the task (must have).
- And/or at least one flowchart for one important function in the program. The student can draw more than one flowchart for different functions.

The students must provide the folder structure and file structures as the followings with **brief description**:



Where the project root folder is named as "groupID _ mini_project_20232" and three subfolders "task_1", "task_2" and "task_3" which contain the related files and should not have any subfolders. "report.docx" is the report file and should be placed in the project root folder. The *groupID* in the folder name should be replaced by the group ID.

IV. Coding styles:

Coding style needs to be consistent throughout the program and follows the GNU style described in the following link: https://www.gnu.org/prep/standards/html_node/Writing-C.html

Shortly, it should be as the followings:

- The structure of the code should be clean and easy to read by using proper indentation, parenthesis, code block, line break and spacing.
- Comments are provided to explain the program more clearly but not to paraphrase the code statement.
- The names of functions and variables must be in English, compact and self-described.
- Hard-coding should be avoided.

Note: The students must NOT use any third-party libraries rather than the C/C++ standard library.

V. The tools:

Editor: Visual studio code (https://code.visualstudio.com/download)

Compiler: gcc or g++ in MinGW-w64 (https://sourceforge.net/projects/mingw-w64/) version 8.1.0

The students should also mention in the report that the program is written in which Operating System (Windows, Linux, MacOS).

VI. Report and submission guidlines:

- The students do the mini project in a group.
- The whole project needs to be organized in a folder as describe in section III.
- The studens must write an English report in a Word file named as "report.docx" which should not exceed 6 A4 pages and should not include the source code. The report must follow IEEE template which is attached in the Team Assignment.
- The content of the report must be:
  - Introduction: Brief description of the program design idea including
    - the folder structure,
    - the source code files (if there are multiple source code files),
    - and the standard libraries used

    Do NOT rewrite this mini project description in the report.
  - Detailed design:
    - introduction of the crucial main data structures defined by students to handle the data (if any),
    - the top-down approach diagrams as mentioned in section III
    - design description of some selected important functions including the function call syntax (function name, argument list and returning value) and inputs/outputs, pre-conditions, post-conditions;
    - the flowcharts as mentioned in section III.

- Results and evaluations: summarising the results of the program execution and its performance.
- Conclusions:
  - briefly conclude what have been done and what have NOT been done;
  - provide a table of group member contribution as the below example:

Student names	Tasks	Percentage of contribution
Nguyen Van A	1, 2.1, 2.2	50%
Nguyen Van B	2.3, 2.4, 3	50%

  If only one student completes all the work and the other do nothing, one can write the percentage of contribution as 100% and 0% respectively.
- References (If any).

- The students must compress the whole project folder in a **zip** file and name it as "groupId_mini_project_20232.zip" for submission. Please take note that it must be a **ZIP** file but NOT any other compression files like .rar.
  - Keep only the source codes, execution files and the Word report file.
  - Unrelated files should be removed before submitting.
  - The "groupID" in the file name must be replace by the group ID, e.g.: if the groupd ID is 20231234_20232345 the submission zip, file should be "20231234_20232345_mini_project_20232.zip"
- Students must submit the zip file above in Team Assignment by the deadline specified there. Do NOT submit via email, Teams chat or any other channels. Only ONE submission per group is required.
- If there are concerns on anything else which is NOT mentioned in this project instruction, the students can contact the lecturer for clarification. It is highly recommended to ask the questions in class Teams rather than private discussion with the lecturer so that every student in the class is informed unless the question is too personal.

VII. Evaluations:

- The mini project is evaluated as below:
  - All the tasks are completed, no run-time error, proper error handling and creative implementation (60%)
  - Clean and reusable design (20%)
  - Good coding style (20%)
  - Clear and well-structure report properly following template and highly consistent with the source code (20%)
  - Improper naming and structure of submission files and folder as specified in Section III (-5%)
- The students must do the project themselves. Do **NOT** copy others' works. The group must keep their work **confidential**. If any two or more groups have the similar source codes and/or reports, all the works of those groups are unacceptable and are considered as they have not submitted yet. It does not matter who copies from whom!
- **No late** submission is allowed.
- Good performance in the mini project can be awarded bonus points in course progress evaluation.
- The group who does not submit the mini project will lose 3 point (over 10) in the course progress evaluation.

---------- END ----------