Group - Mini project

PM2.5 dust sensor simulation and data analysis

I. Description:

Students need to write programs in C or C++ with the appropriate functions and data structures to simulate PM2.5 dust sensors which measure the concentration of dust particles with the size < 2.5 microns in ambient air. Main sensor specifications are:

- Measurement range: $0 \div 800\ \mu g/m^3$
- Resolution: $0.1\ \mu g/m^3$

The required tasks include:

1. Task 1:

Write a program which allow the user to provide the number of sensors, sampling time and measurement duration by using command-line statement to generate simulation data. The format of the command-line statement is as the folllowings:

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

Where:

- `dust_sim`: is the name of the compiled program file.
- `-n [num_sensors]` are a pair of input arguments to provide the number of sensors, `[num_sensors]` must be replaced by a specific positive integer value. If only one of these two appears in the command-line statement, error message must be delivered. If both of these two are not given in the statement, the default number of sensor is used and it is **1 (one)**.
- `-st [sampling]` are a pair of input arguments to provide the sampling time, [`sampling]` must be replaced by a specific positive integer value **in second**, the smallest sampling time allowed is **10 second**. If only one of these two appears in the command-line statement, error message must be delivered. If both of these two are not given in the statement, the default sampling time is used and it is **60 seconds**.
- `-si [interval]` are a pair of input arguments to provide the simulation/measurement duration, `[interval]` must be replaced by a specific positive integer value **in hour**, the smallest duration allowed is **1 hour**. If only one of these two appears in the command-line statement, error message must be delivered. If both of these two are not given in the statement, the default duration is used and it is **24 hours**.

The simulation data generated include the sensor identification number (sensor id), (simulated) measurement timestamp, and (simulated) sensor value. The starting time of the simulatiuon is identified by subtracting the current time in the system (computer time) with the simulation duration. The timezone should be default, it is usually the local time; student do not need to use any function or argument to change the timezone.

- The sensor id is generated in the range of 1 to `num_sensors`, where `num_sensors` is the number of sensor that the user provided in the command-line statement, e.g. if `num_sensors = 10` the program will create 10 sensors with the ids are 1, 2, 3, …, 10.
- The measurement timestamp (simulated) needs to be written in the format of YYYY:MM:DD hh:mm:ss, where:
  - YYYY – year, MM – month, DD – day.
  - hh – hour, mm – minute, ss – second.

  E.g: 2024:05:01 08:30:02
- The measurement value (simulated) is generated randomly with the precision of 1 digit after decimal point.

NOTE: the simulation time (duration and timestamp) is not real-time, it is just computed in simulation, thus, the student do not use time delay functions such as sleep() or loop to create time delay.

The data generated needs to be stored in a file named "dust_sensor.csv"; if the file exists, the program can override the old file. This data file follows the CSV (comma-separated values) format, each field is separated by a comma. CSV file format can be referred in the url: https://www.ietf.org/rfc/rfc4180.txt. The data file needs to be in the same folder as the program file.

Example: Let a user write the command-line statement:

```
C:\\dust_sim –n 3 –st 60 –si 10
```

- Asumming that the user run the command line statement at 2024:11:02 10:00:00, the starting time of the simulation is at 2024:11:02 00:00:00. Both the starting time and the execution time are included.
- Data in the file "dust_sensor.csv" are as the following:

```
id,time,value
1,2024:11:02 00:00:00, 50.1
2,2022:11:02 00:00:00,24.2
3,2024:11:02 00:00:00, 200.5
1,2024:11:02 00:01:00,100.2
2,2024:11:02 00:01:00,55.4
3,2024:11:02 00:01:00,160.9
…
1,2024:11:02 10:00:00,120.2
2,2024:11:02 10:00:00,90.4
3,2024:11:02 10:00:00,351.0
```

The first line "id,time,value" is the header line.

2. Task 2:

Students need to write a program to process a csv with the same format as in task 1. The program must be executed with the following command-line statement:

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

Where:

- `dust_process`: is the compiled program file
- `[data_filename.csv]` is the csv file consisting of the dust sensor data. In case the user does not provide the filename but type only `dust_process`, the program should use the default filename as "dust_sensor.csv" and look for the data file in the same folder of the program file.

For example: `dust_process dust_sensor_ee3490e.csv`

The program must be able to process upto 10000 data points, i.e.: the input file `data_filename.csv` may consist of upto 10000 data lines. The outcomes of data processing are as the following tasks. All tasks must be performed with the same executation file.

a. Task 2.1:

It is assumed that the dust concentration in the monitored environment can only be in the range of $3 \div 550.5\ \mu g/m^3$. Thus, the valid sensor values must be within this range, and any values which are not in this range are outliers. The program needs to look for the invalid sensor values, i.e. outliers in the csv file and stored in another csv file named "dust_outlier.csv". An example of this file content is shown below:

```
number of outliers: 3
id,time,value
1,2024:11:02 00:00:00,2.1
3,2024:11:02 19:03:00,-1.0
3,2024:11:02 21:06:00,560.2
```

In which, the first line is written as "number of outliers: X" with X is the number of outliers found in the data file, in this example X = 3. If there is no invalid value, X is 0 and there is no data line in this file.
The valid values are stored in another csv file named "dust_valid.csv" which has the same format as the original data file.

NOTE: The valid sensor values are used to perform the rest of the tasks from task 2.2 onwards.

b. Tasks 2.2:

The dust concentration can be converted into Air Quality Index (AQI) as the followings:

Table 1. Air quality index and polution level

Concentration c $[\mu g/m^3]$	**AQI**	**Pollution level**	**Pollution code**
$0 \leq c < 12$	0 ÷< 50	Good	A
$12 \leq c < 35.5$	50 ÷ <100	Moderate	B
$35.5 \leq c < 55.5$	100 ÷ <150	Slightly unhealthy	C
$55.5 \leq c < 150.5$	150 ÷ <200	Unhealthy	D
$150.5 \leq c < 250.5$	200 ÷ <300	Very unhealthy	E
$250.5 \leq c < 350.5$	300 ÷ <400	Hazardous	F
$350.5 \leq c \leq 550.5$	400 ÷ 500	Extremely hazardous	G

The program needs to calculate the average dust concentration per hour, e.g. average dust concentration at 2023:05:08 02:00:00 is the average value of all the concentration values from 2023:05:08 01:00:00 to 2023:05:08 01:59:59. It also identify the AQI and pollution level with respect to that average value. The resusts should be store in a file named "dust_aqi.csv" with the same format as the below example:

```
id,time,value,aqi,pollution
1,2024:10:28 23:00:00,8.5,35,Good
2,2024:10:28 23:00:00,64.8,155,Unhealthy
3,2024:10:28 23:00:00,197.2,247,Very unhealthy
4,2024:10:28 23:00:00,239.7,289,Very unhealthy
5,2024:10:28 23:00:00,134.9,192,Unhealthy
1,2024:10:29 00:00:00,42.9,118,Slightly unhealthy
2,2024:10:29 00:00:00,89.7,168,Unhealthy
3,2024:10:29 00:00:00,249.4,299,Very unhealthy
…
```

c. Tasks 2.3:

Identify the maximum (max), minimum (min), the average concentration(mean), and the median values of each sensor's measurements over the simulation duration. The results must be stored in a file named "dust_summary.csv" with the same format as the below example.

The time of the max and min values are the earliest timestamps these values appear in the input file. The time of the mean and median value is the simulation time interval.

```
id,parameter,time,value
1,max,2024:10:29 14:17:39,529.0
1,min,2024:10:29 19:01:39,3.1
1,mean,10:00:00,71.03
1,median,10:00:00,46.5
2,max,2024:10:29 21:47:39,546.2
2,min,2024:10:29 12:59:39,3.0
2,mean,10:00:00,36.36
2,median,10:00:00,8.9
3,max,2024:10:29 21:41:39,508.1
3,min,2024:10:29 21:51:39,3.6
3,mean,10:00:00,88.51
3,median,10:00:00,48.2
…
```

Hint: you may need to sort the sensor values from smallest to largest to find the median.

d. Task 2.4:

Based on the outcomes of task 2.2, the program needs to calculate total number of hours for each pollution level at each sensor and stored the results in a file named "dust_statistics.csv" with the same format as the following example

```
id, pollution,duration
1,Good,2
1, Moderate,1
1, Slightly unhealthy,3
1,Unhealthy,0
1,Very unhealthy,2
1, Hazardous,2
1, Extremely hazardous,0
2,Good,1
2, Moderate,2
2, Slightly unhealthy,0
2,Unhealthy,4
2,Very unhealthy,1
2, Hazardous,0
2, Extremely hazardous,1
…
```

3. Task 3:

It is assumed that the users need to send the output data of task 2.2 over a communication protocol. The data packet is a byte array which must be created as the below frame structure

Start byte	Packet Length	ID	Time	PM2.5	AQI	Pollution code	Checksum	Stop byte
0xAA (1 byte)	1 byte	1 byte	4 bytes	4 bytes	2 byte	1 byte	1 byte	0xFF (1 byte)

Where:

- Start byte (1 byte) is the first byte in the packet and always has the value of 0xAA.

- Stop byte (1 byte) is the last byte in the packet and always has the value of 0xFF.
- 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.
- PM2.5 is the PM2.5 dust concentration value which is a 4-byte real number represented with IEEE 754 single precision floating-point standard.
- AQI is the air quality index and is a 2-byte integer.
- Pollution code is the ASCII code with repect to the character representing pollution code of the AQI value in Table 1.
- 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, PM2.5, AQI, pollution code]

All the numbers (integer and real) are represented as **little-endian**.

If the user executes the following command-line statement:

```
C:\\ dust_convert [data_filename.csv] [hex_filename.dat]
```

Where

- `[data_filename.csv]` is an input file which has the same format described in task 2.2.
- `[hex_filename.dat]` is the output file in text format with the extension of ".dat".

The users must provide both the two filenames, otherwise it is an invalid statement.

The program should:

- Read each line of the input file, i.e.: `[data_filename.csv]`
- Convert each data line into the data packet as described above, each byte is separated by a space character and represented as hex number
- Write each packet in one line in the output file, i.e. `[hex_filename.dat]`
- Overide the output file `[hex_filename.dat]` if it has been existed.
- Be able to process at least 10000 data ponts which means that the input file `data_filename.csv` may consist of at least 10000 lines.

For example:

```
C:\\dust_convert dust_aqi.csv hex_packet_ee3490e.dat
```

A line of "2,2024:09:18 20:06:11,499.7,475,Extremely hazardous" exited in the file named "dust_aqi.csv" is converted into

AA 10 02 43 D0 EA 66 9A D9 F9 43 DB 01 47 B9 FF

II. Other technical requirements:

The execution time of each task (1, 2, 3) should not exceed **30 seconds**.

The program needs to store any run-time errors occurring in log files, there must be 3 different log files for 3 tasks in section I. They are named as **task1.log**, **task2.log** and **task3.log**. 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".

- "dust_sensor.csv" file is existing and is a read-only file. The error message can be "Error 03: dust_sensor.csv 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` 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 csv file format"
- The user types the wrong command-line format, e.g.: one or both the two filenames are missing. The error message can be "Error 03: invalid command"
- The input file contains wrong data:
    - All the data fields in one line are blank, e.g.: ", , "
    - Id is blank or invalid, e.g.: "-1, 2023:05:08 00:00:00, 50.1"
    - Time is blank or invalid, e.g.: "1,2023:05:08 00:00:, 50.1"
    - Concentration value is blank, e.g.: "1,2023:05:08 00:00:00, "

  The error message must include the line number in the input file in which the error happens, i.e.: "Error 04: data is missing 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 when perform task 2.
- The input file contains duplicated data:
    - The input file contains two or more lines which have identical id and time.

  The error message must include the line number in the input file in which the error happens, i.e.: "Error 05: data is duplicated at line X and Y" where X and Y 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 lines and are numbered from 1 (one).
  The program should process non-duplicated data lines and only the first line with duplicated data but ignore the others line with duplicated data.
- The input file has more than 10000 data lines, the program should process up to 10000 data lines and ignore the rest. It should also give an error message as "Error 06: input file is too large"

3. Errors may happen in task 3:

- 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` 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 csv 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 input file contains wrong data:
    - All the data fields in one line are blank, e.g.: ", , , , "
    - Id is blank or invalid, e.g.: "**-1**,2024:09:18 20:06:11,11.1,46,Good"
    - Time is blank or invalid, e.g.: "1,2024:09:18 20:06:**71**,11.1,46,Good"
    - Concentration value is blank, e.g.: "1,2024:09:18 20:06:11, ,46,Good"
    - Aqi value is blank or invalid, eg.: "1,2024:09:18 20:06:11,11.1, ,Good"
    - Concentration value and aqi value are not consistent as described in table 1, e.g.: "-1,2024:09:18 20:06:11,**11.1,146**,Good"

  The error message must include the line number in the input file in which the error happens, i.e.: "Error 04: data is missing 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 when perform task 2.

- The input file contains duplicated data:
    - The input file contains two or more lines which have identical id and time.

  The error message must include the line number in the input file in which the error happens, i.e.: “Error 05: data is duplicated 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 process non-duplicated data lines and only the first line with duplicated data but ignore the others line with duplicated data.
- The input file has more than 10000 data lines, the program should process up to the first 10000 data lines and ignore the rest. It should also give an error message as “Error 06: input file is too large”
- The output file, e.g.: `hex_filename.dat` is existing and cannot be overriden. The error message can be “Error 07: cannot override the hex file”

4. Other cases:

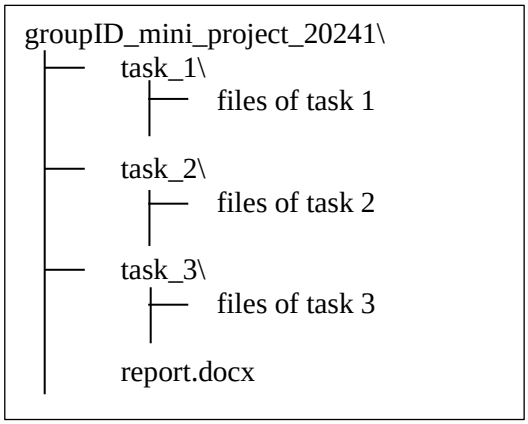
The students can suggest more errors which may happen but not be listed above. Those errors should be stored in the respective log file and described in the report.

III. Program design:

The students should provide the clear analysis and proper design idea of the programs.
It is required to draw the top-down approach diagram to illustrate structure of the programs and the relationship between the functions in the program together with a brief description in the report.
The students also need to draw the following flowcharts:

- At least one flowchart of the overall program in all the tasks.
- And at least one flowchart for one important function in the program of each task.

The students must provide the folder structure and file structures with brief description as the following format:



Where the project root folder is named as “groupID_mini_project_20241” and three subfolders “task_1”, “task_2” and “task_3” are used to contain the related file for each task respectively. “report.docx” is the report file and should be placed in the project root folder. The *groupID* in the filename should be replaced by the group ID. There should not be any subfolders in “task_1”, “task_2” and “task_3” folders.

Important data structures used and defined in projects should also be describe and explained clearly.

Note: The students can use only the C/C++ standard library but not any third-party libraries.

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 (4 spaces), parenthesis, code block, line break and spacing.
- Comments are provided for the programs, the functions, and important statements to explain the program more clearly but do not paraphrase the code statement.
- The names of functions and variables must be in English, compact and self-described with consistent naming convention.
- Hard-coding should be avoided by defining constants where possible.

V. The tools:

Editor: Visual studio code (https://code.visualstudio.com/download)
Compiler: gcc or g++ in MinGW-w64: https://github.com/niXman/mingw-builds-binaries/releases/download/14.2.0-rt_v12-rev0/x86_64-14.2.0-release-posix-seh-ucrt-rt_v12-rev0.7z
The students should also mention in the report that the program is written in which Operating System (Windows, Linux, MacOS).
Students should try to do yourselves using the above tools only for practicing. Do NOT rely on the AI tools like ChatGPT or Copilot.

VI. Report and submission guidlines:

- The students need to do the mini project in a group.
- The whole project needs to be organized in a folder structure as described 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 of main idea: Brief description of the program design, the top-down approach diagram, main data structures, the folder structure, the source code files (if there are multiple source code files) and the standard libraries used.
  - Detailed design: introduction of the new key data structures used and/or defined by students to handle the data (if any), description of the function designs including the function call syntax (function name, argument list and returning value) and inputs/outputs, pre-conditions, post-conditions; flowcharts as mentioned in section III. If there are many functions written in the program, students can select a few important ones to introduce.
  - Results and evaluations: present 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).

  Do **NOT** rewrite this project instruction or copy source code in the report.
- The students must compress the whole project folder in a zip file and name it as "groupId_mini_project_20241.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 and the Word report file in the submission. Do not include or any unrelated files in the submission.
- The execution files, data files and unrelated files should be removed before submitting.
- The "groupID" in the file name must be replace by the group ID, e.g.: "20221234_20222345_mini_project_20222.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, fast execution time, 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 Secion III (-5%)
- The students must do the project themselves. Do NOT copy others' works or any other sources. The group should 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 plagiarism. It does not matter who copies from whom.** Codes or report copied from other sources or AI tools are also considered plagiarism.
- No late submission is allowed. Let's submit whatever the group will have done by the deadline specified in Teams assignment.
- Good performance in the mini project can be awarded bonus points in course progress evaluation. The students can be given maximum bonus point if all the three tasks are done and pass a few test cases.
- The students may be questioned individually and randomly to verify if the projects are done by the students themselves.
- The group who does not submit the mini project or plagiarise will lose 3 points (over 10) in the course progress evaluation. The student in the group has little contribution will also lose some points.

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