PM2.5 dust sensor simulation and data analysis

with C language

Trần Đăng Khoa–20212413 & Đỗ Đức Toàn-20212422

# I. INTRODUCTION

**Overview about PM2.5 dust sensor :**

It is used to collect air formaldehyde, ammonia, benzene and total volatile organic compound (TVOC) gas. PM2.5 dust sensor has many applications in life, the most typical is that this sensor can help us perceive the quality of the surrounding air. The dust sensor's operating range is adjustable or different depending on the user's purpose. It is used for indoor and outdoor environmental monitoring institutions, occupational health monitoring institutions, industrial and mining enterprises, health and epidemic prevention institutions, and scientific research institutes to collect various toxic and harmful gases.

**Program idea:**

* Using top-down approach to design the program.
* According to the instructions, we separate the three tasks task 1 , task 2 and task 3 into three separate files.

**The main content of the implementation:**

**Task 1:**

* Create a file named “dust\_sensor.csv” that allow user to get data randomly
* Check the command line
* Using time(NULL) and other time related functions to generate starting and ending time, then use the rand() function to get random values.
* Error will be printed in a file named task1.log

**Task 2:**

* Task 2.1: Get data from the created file of task 1 and check that the dust concentration values are in the range of 5 ÷ 550 or not. If the values are not in that range, print into the a csv file named “dust\_outlier.csv”
* Task 2.2: Read each valid sensor. In each sensor, get the dust concentration values being in the same hour and calculate the average of them. After that, find the AQI and pollution level with respect to that average value and printf into a file named “dust\_aqi.csv”
* Task 2.3: Read each valid sensor. In each sensor, compare each dust concentration values to find the maximum, minimum and calculate the average concentration values over all the time then printf into a file named “dust\_summary.csv”
* Task 2.4: Get the data related to AQI from task 2.2, count the number of hours for each pollution level at each sensor and stored the results in a file named “dust\_statistics.csv”
* Error will be printed in a file named task2.log

**Task 3**

* Read the input file and store the data in a struct to handle
* Separate each byte by a space character and represent it as a hex number.
* Error will be printed in a file name task3.log

**Objectives of the study**

* Make a program in C with the appropriate functions and data structure to simulate PM2.5 dust sensor which measures the concenstration of dust particle with the size < 2.5 microns in ambient air.

**Structure diagram of the project**

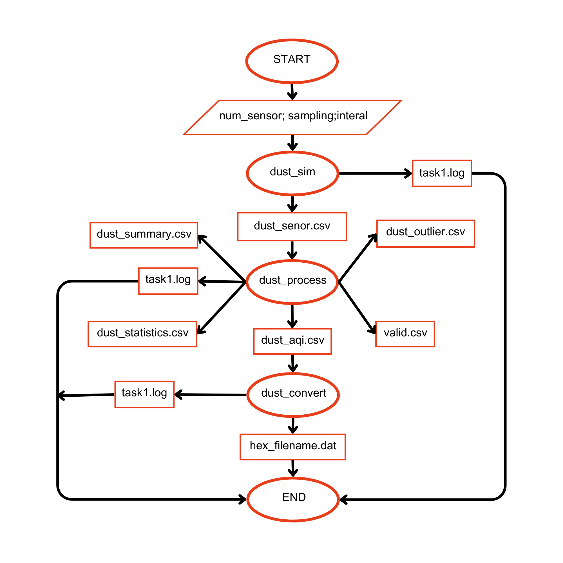
**Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, biểu đồ

Mô tả được tạo tự động**

**Fig.1.** Structure of project

# II. Design program

As observed in Figure above, the research is divided into 3 main parts (task 1, task 2, task 3). We will follow the flowchart in Figure below, representing the main idea that covers the whole study and looks at the design process piecemeal.



**Fig. 2.** Main idea of project

## **A. Task I – create and add data into a CSV file**

In this task, we will design a program that allows users to use command lines to add data into a CSV file. If the command is in the correct format, it will create a CSV file named dust\_sensor.csv and randomly create data inner it. If the command is incorrect, the program will return an error report in a log file named “task1.log”.

**A diagram of a algorithm

Description automatically generated**

**Fig. 3.** Example about operation

of one special function in this task

* **Function Name: main**
* Argument List: uint8\_t argc, char \*\*argv
* Returning Value: int
* Description: The main function is the entry point of the program. It performs the main logic of the program, which involves parsing command line arguments, generating data, and writing the data to a CSV file. It utilizes the get\_argument and log\_error functions.
* Inputs:

argc: The number of command line arguments passed to the program.

argv: A pointer to an array of strings, where each string represents a command line argument.

* Outputs:

Returns an integer value:

0: If the program execution is successful.

1: If there is an error in the program execution.

* Pre-conditions:The argc should be greater than 0.The argv should contain valid command line arguments.The "task1.log" file should be writable.
* Post-conditions:

The program may create or overwrite the "task1.log" file and write error messages to it.

The program may create or overwrite the "dust\_sensor.csv" file and write data to it.

The program returns 0 if the execution is successful, or 1 if there is an error.

* **Function Name: get\_argument**
* Argument List: uint8\_t argc, char \*\*argv
* Returning Value: int
* Description:The function get\_argument takes in command line arguments as input and parses them to extract specific values. It is designed to handle a specific set of command line options: -n, -st, and -si.
* Inputs:

argc: The number of command line arguments passed to the program.

argv: A pointer to an array of strings, where each string represents a command line argument.

* Outputs:

Returns an integer value:

0: If the parsing is successful.

error\_1: If there is an error in the argument format (e.g., missing value or invalid option).

error\_2: If there is an error converting the argument value to a numeric format.

* Pre-conditions:

The argc should be greater than 0.

The argv should contain valid command line arguments.

* Post-conditions: The variables num\_sensors, sampling, and interval are updated based on the parsed command line arguments. If the command line arguments are not provided or are invalid, default values are assigned to num\_sensors, sampling, and interval. The function returns 0 if the parsing is successful. Otherwise, it returns an error code.
* **Function Name: log\_error**
* The function writes the error message to the log file.

## **B. Task II – process the data**

With the CSV file created in task 1,we will create a program to process the required data from task2.1 to task 2.4:

* **Function Name: main**
* Argument List: int argc, char \*\*argv
* Returning Value: int
* Description: The main function is the entry point of the program. It calls several task-specific functions: task2\_1, task2\_2, task2\_3, and task2\_4.
* Inputs:

argc: The number of command line arguments passed to the program.

argv: A pointer to an array of strings, where each string represents a command line argument.

* Outputs: a file named *“dust\_sensor.csv”*

Returns an integer value:

0: If the program execution is successful.

1: If there is an error in the program execution.

* Pre-conditions:

The argc should be greater than or equal to 2.

The argv should contain valid command line arguments.

The "task1.log" file should be writable.

The input CSV file specified in argv[1] should exist and be readable.

* Post-conditions:

The program may create or overwrite the "task1.log" file and write error messages to it.

The program reads data from the input CSV file and stores it in a data structure.

The program performs various tasks (task2\_1, task2\_2, task2\_3, task2\_4) on the data and prints the results to the screen.

The program returns 0 if the execution is successful, or 1 if there is an error.

* **Function Name: task2\_1**
* Argument List: None
* Returning Value: int
* Description: The task2\_1 function performs a specific task related to processing data from the sensors array. It separates the data into two categories: valid data and invalid data based on a specified condition. It then stores the invalid data in a files: "dust\_outlier.csv" for invalid data.
* Inputs: None
* Outputs: a file named *“dust\_outlier.csv”*

Returns an integer value:

0: If the task execution is successful.

1: If there is an error in the task execution.

* Pre-conditions:

The sensors array should be populated with data.

The g\_n\_data,valid\_data\_sensors, invalid\_data\_sensors, n\_valid\_data, and n\_invalid\_data variables should be properly initialized.

* Post-conditions:

The valid\_data\_sensors and invalid\_data\_sensors arrays are populated with the corresponding data.

The number of valid data points (n\_valid\_data) and invalid data points (n\_invalid\_data) are updated.

If there are any file opening or writing errors, appropriate error messages are printed to the console, and the function returns 1.

* **Function Name: task2\_2**
* Argument List: None
* Returning Value: uint32\_t
* Description: The task2\_2 function performs a specific task related to processing data from the valid\_data\_sensors array. It calculates the average value of each sensor's data for each hour, determines the corresponding Air Quality Index (AQI) and pollution level, and stores the results in the aqi\_data\_sensor array. It also sorts the aqi\_data\_sensor array based on the AQI values. Finally, it writes the sorted data to the "dust\_aqi.csv" file.
* Input: None
* Outputs: a file named *“dust\_aqi.csv”*

Returns an unsigned 32-bit integer value:

0: If the task execution is successful.

1: If there is an error in the task execution.

* Pre-conditions:

The valid\_data\_sensors array should be populated with valid data.

The g\_max\_id, n\_valid\_data, aqi\_data\_sensor, and g\_n\_aqi variables should be properly initialized.

* Post-conditions:

The aqi\_data\_sensor array is populated with the calculated AQI, pollution level, and other relevant data.

The aqi\_data\_sensor array is sorted based on the AQI values.

The "dust\_aqi.csv" file is created or overwritten and populated with the sorted data.

If there are any file opening or writing errors, appropriate error messages are printed to the console, and the function returns 1.

* **Function Name: task2\_3**
* Argument List: None
* Returning Value: int
* Description: The task2\_3 function performs a specific task related to processing data from the valid\_data\_sensors array. It calculates summary statistics for each sensor, including the maximum value, minimum value, and mean value. It stores the results in the summary\_data\_sensor array. Finally, it writes the summary data to the "dust\_summary.csv" file.
* Inputs: None
* Outputs: a file named “dust\_summary.csv”

Returns an integer value:

0: If the task execution is successful.

1: If there is an error in the task execution.

* Pre-conditions:

The valid\_data\_sensors array should be populated with valid data.

The g\_max\_id, n\_valid\_data, summary\_data\_sensor, min\_ts, and max\_ts variables should be properly initialized.

* Post-conditions: The summary\_data\_sensor array is populated with the summary statistics for each sensor.The "dust\_summary.csv" file is created or overwritten and populated with the summary data.If there are any file opening or writing errors, appropriate error messages are printed to the console, and the function returns 1.
* **Function Name: task2\_4**
* Argument List: None
* Returning Value: int
* Description: The task2\_4 function performs a specific task related to processing data from the aqi\_data\_sensor array. It calculates the duration of each pollution level for each sensor and stores the results in the statistic\_data\_sensor array. Finally, it writes the statistical data to the "dust\_statistics.csv" file.
* Inputs: None
* Outputs: a file named “dust\_statistics.csv”

Returns an integer value:

0: If the task execution is successful.

1: If there is an error in the task execution.

* Pre-conditions:

The aqi\_data\_sensor array should be populated with valid data.

The g\_max\_id, g\_n\_aqi, and statistic\_data\_sensor variables should be properly initialized.

* Post-conditions:

The statistic\_data\_sensor array is populated with the duration of each pollution level for each sensor.

The "dust\_statistics.csv" file is created or overwritten and populated with the statistical data.

If there are any file opening or writing errors, appropriate error messages are printed to the console, and the function returns 1.

## **C. Task III – converting the data**

* **Function Name: main**
* Argument List: int argc, char \*argv[]
* Returning Value: int
* Description: The main function is the entry point of the program. It takes command-line arguments argc (argument count) and argv (argument vector) to specify the input and output file names.
* Inputs:

argc: An integer representing the number of command-line arguments.

argv: An array of strings representing the command-line arguments.

* Outputs: the output file [hex\_filename.dat]

Returns an integer value:

0: If the program execution is successful.

1: If there is an error in the program execution.

* Pre-conditions:

The program expects at least 3 command-line arguments (including the program name).

The input and output file names should be valid and accessible.

The log\_error function should be defined and handle appropriate error codes.

The *read\_input\_filename, floatToIEEE754, check\_valid\_time, and handle\_string* functions should be defined and handle their respective tasks.

* Post-conditions: The output file specified by file\_name\_output is created or overwritten and populated with the formatted output data.

If there are any file opening or writing errors, appropriate error messages are printed to the console, and the program returns 1.

* **Function Name: handle\_string**
* Argument List: uint32\_t value, char \*pdest
* Returning Value: void
* Description: The handle\_string function takes an unsigned 32-bit integer value and a pointer to a character array (pdest). It converts the value to a hexadecimal string representation and formats it with spaces after every 2 characters. The formatted string is then stored in the character array pdest.
* Inputs:

value: An unsigned 32-bit integer value.

pdest: A pointer to a character array where the formatted string will be stored.

* Outputs: None. The result is stored in the pdest character array.
* Pre-conditions:

The pdest character array should have sufficient space to accommodate the formatted string.

The value should be a valid unsigned 32-bit integer.

* Post-conditions: The pdest character array contains the formatted string representation of the value with spaces after every 2 characters.
* **Function Name: floatToIEEE754**
* Argument List: float value
* Returning Value: uint32\_t
* Description: The floatToIEEE754 function takes a single-precision floating-point value (value) and converts it to its corresponding 32-bit unsigned integer representation in IEEE 754 format. It uses a union of a float and a uint32\_t to perform the conversion.
* Inputs:value: A single-precision floating-point value to be converted.
* Outputs: Returns a 32-bit unsigned integer representing the IEEE 754 format of the input value.
* Pre-conditions: The input value should be a valid single-precision floating-point value.
* Post-conditions: The function returns the 32-bit unsigned integer representation of the input value in IEEE 754 format

# III. Result

## **A. Task I – create and add data into a CSV file**

We can use the command line to create the CSV file.

Example of the command line argument:



The result of the above line will return in a CSV file and have the format as the figure …

A screenshot of a computer

Description automatically generated**Fig. 4.** Example of the result in the CSV file

If error, the result will return the name of the error in a log file named **task1.log**.

The program runs quite quickly and generates a lot of data. With this program, we can simulate the values of dust sensor quickly with just one command with the exact display time at the time of running the program..

For more options to be done with the created data, we can find them in the part B of task 2 – process the data.

## **B. Task II – process the data**

The data processing:

C:\\dust\_process dust\_sensor.csv

After run this program, we will have 4 file CSV:

+dust\_summary.csv

+dust\_statistics.csv

+dust\_ourlier.csv

+dust\_aqi.csv

A screen shot of a computer

Description automatically generated

**Fig.5**. the result in the **dust\_summary.csv**

A screenshot of a computer program

Description automatically generated

**Fig.6.** the result in the **dust\_statistics.csv**

A screenshot of a computer

Description automatically generated

**Fig.7.** the result in the **dust\_ourlier.csv**

A screenshot of a computer

Description automatically generated

**Fig.8.** the result in the **dust\_aqi.csv**

If error, the result will return the errors in a log file named **task2.log**.

When executing the above command correctly, the program will create 4 files CSV at the same time with the results displayed as described in each task.

Finally at task 3, we convert the input data from task 2 to hexadecimal in the output data file

## **C. Task III – convert data**

When we run the program with the command format described in task 3, we get the following output file:

A screenshot of a computer screen

Description automatically generated

**Fig.9.** the result in the output file

The same with 2 task above,if error, the result will return the errors in a log file named **task3.log**.

# IV. Conclusion

Because it has been tested many times, the above program has been fully programmed and is suitable for the given requirement of simulating the principles of a dust sensor. Although this C program has a lot of steps and quite a long code, it can handle the operation with quick results and high accuracy.

Below is the task allocation table of our team.

|  |  |  |
| --- | --- | --- |
| Name | Tasks | Percentage of contribution |
| Trần Đăng Khoa | 2.4, 3 | 45% |
| Đỗ Đức Toàn | 1, 2.1, 2.2, 2.3 | 55% |

References

[1] Hoang D. Chinh, *“mini\_project\_20212-2.pdf”*, HUST

[2] W3school, *C programming language,*

<https://www.w3schools.com/>