# Mini project

## Illuminance sensor simulation and data analaysis

## I. Description:

Students need to write a program in C or C++ with the appropriate functions and data structure to simulate illuminance sensor operation which measures the illuminance of indoor space. Main sensor specifications are:

- Measurement range: $0.1 \div 40000\ Lux$
- Resolution: $0.01\ Lux$

The tasks to do 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:\\lux_sim -n [num_sensors] -st [sampling] -si [interval]
```

Where:

- `lux_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 1 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, i.e. the fist timestamp, is identified by subtracting the current time in the system (computer time when the program is executed) with 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: 2023:11:01 08:30:02
- The measurement value (simulated) is generated randomly with the precision of 2 digits after decimal point.

**Notice:** 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 "lux_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:\\lux_sim -n 3 -st 60 -si 10
```

- Asumming that the user run the command line statement at 2023:11:11 10:00:00, the starting time of the simulation is at 2023:11:11 00:00:00. Both the starting time and the `execution time are included.`
- Data in the file "`lux_sensor.csv`" are as the following:

```
id,time,value
1,2023:05:08 00:00:00, 50.01
2,2022:05:08 00:00:00,24.02
3,2023:05:08 00:00:00, 200.05
1,2023:05:08 00:01:00,100.12
2,2023:05:08 00:01:00,55.34
3,2023:05:08 00:01:00,160.49
…
1,2023:05:08 10:00:00,120.52
2,2023:05:08 10:00:00,90.40
3,2023:05:08 10:00:00,351.00
```

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:\\lux_process [data_filename.csv] [location.csv]
```

Where:

- `lux_process`: is the compile program file
- `[data_filename.csv]` is the csv file consisting of the light sensor data.
- `[location.csv]` is the csv file consisting of the location code for each sensor with the format as an example below:

```
id,location
1,2
2,3
3,4
4,10
5,2
…
9,5
```

Where the id column contains all the sensor id available in `[data_filename.csv]` and location is the location code as described in Table 1.

For example: `C:\\lux_process lux_sensor_ee3490e.csv location.csv`

If none of the csv files or only one csv file is provided, the command is invalid.

The program must 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. The outcomes of data processing are as the following tasks. All sub-tasks of task 2 must be performed at once when the program is executed.

If the sensor in data file is not included in the location file, e.g. data file has 9 sensors with ids from 1 to 9 but the location file has only 7 sensor ids 1, 2, 5, 6, 7, 8, 9, the location code of sensor 3 and 4 are 0 (unknown location).

*Table 1. Type of location with related activities and the required lux level*

| Type of area with related activity | Code | Min required lux | Max required lux |
|---|---|---|---|
| Unknown | 0 | NA | NA |
| Public areas with dark surroundings | 1 | 20 | 50 |
| Simple orientation for short visits | 2 | 50 | 100 |
| Areas with traffic and corridors - stairways, escalators and travelators - lifts – storage spaces | 3 | 100 | 200 |
| Working areas where visual tasks are only occasionally performed | 4 | 100 | 150 |
| Warehouses, homes, theaters, archives, loading bays | 5 | 150 | 250 |
| Coffee break room, technical facilities, ball-mill areas, pulp plants, waiting rooms, | 6 | 200 | 400 |
| Easy office work (e.g. photocopy area) | 7 | 250 | 350 |
| Class rooms | 8 | 300 | 500 |
| Normal office work, PC work, study library, groceries, show rooms, laboratories, check-out areas, kitchens, auditoriums | 9 | 500 | 700 |
| Supermarkets, mechanical workshops, office landscapes | 10 | 750 | 850 |
| Detailed drawing work, very detailed mechanical works, electronic workshops, testing and adjustments | 11 | 1500 | 2000 |
| Performance of visual tasks of low contrast and very small size for prolonged periods of time | 12 | 2000 | 5000 |
| Performance of very prolonged and exacting visual tasks | 13 | 5000 | 10000 |
| Performance of very special visual tasks of extremely low contrast and small size | 14 | 10000 | 20000 |

a. Task 2.1:

It is assumed that the lux level in the monitored environment can only be in the range of $1 \div 30000\ lux$. 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 check for the invalid sensor values, i.e. outliers in the `[data_filename.csv]` file and stored in a csv file named "lux_outlier.csv". An example of this file content is shown below. If "lux_outlier.csv" is already existing, it should be overwritten with the new one. The first line is written as "number of outliers: X" with X is the number of outliers filtered out from the data file, in this example X = 3.

```
number of outliers: 3
id,time,value
1,2023:11:11 00:00:00,0.08
3,2023:11:11 19:03:00,-1.01
3,2023:11:11 21:06:00,35690.22
```

**Only the valid sensor values are used to perform the rest of the tasks from task 2.2 onwards.**

b. Tasks 2.2:

The lux level can be used to identify the light condition in the monitored area as below:

- If lux value < Min required lux: dark
- If lux value > max required lux: bright
- If min required lux ≤ lux value ≤ max required lux: good

Where min required lux and max required lux are given in table 1 for each type of location/area. If the location is **unknown**, the condition should be "NA" (not available)

The program needs to calculate the average lux level per hour, e.g. average lux level at 2023:11:11 02:00:00 is the average value of all the lux values from 2023:11:11 01:00:00 to 2023:11:11 01:59:59. It also identifies the lux condition with respect to that average value. The resusts should be store in a file named "lux_condition.csv" with the same format as the below example:

```
id,time,location,value,condition
1,2023:11:11 01:00:00,1,30.51,good
2,2023:11:11 01:00:00,2,400.03,bright
3,2023:11:11 01:00:00,10,200.00,dark
4,2023:11:11 01:00:00,0,240.00,NA
1,2023:11:11 02:00:00,1,101.02,bright
2,2023:11:11 02:00:00,2,75.02,good
3,2023:11:11 02:00:00,10,812.05,good
4,2023:11:11 02:00:00,0,2.05,NA
…
```

c. Tasks 2.3:

Identify the maximum (max), minimum (min) and the average lux values over all the time (mean) measured by each sensor. The results must be stored in a file named "lux_summary.csv" with the same format as the below example:

```
id, parameter,time,value
1, max,2023:11:11 08:30:00,350.80
1, min,2023:11:11 09:31:03,5.61
1, mean,10:00:00 ,200.54
2, max,2023:11:11 08:35:00,300.83
2, min, 2023:11:11 09:32:03,15.61
2, mean,10:00:00, 110.55
3, max, 2023:11:11 09:05:02,120.67
3, min, 2023:11:11 09:21:03,20.81
3, mean,10:00:00,70.59
…
```

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

**3. Task 3:**

It is assumed that the users need to send and receive the output data of task 2.2 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 | Location | Lux | Checksum | Stop byte |
|---|---|---|---|---|---|---|---|
| 0xA0 (1 byte) | 1 byte | 1 byte | 4 bytes | 1 byte | 4 bytes | 1 byte | 0xA9 (1 byte) |

Where:

- Start byte (1 byte) is the first byte in the packet and always has the value of 0xA0.

- Stop byte (1 byte) is the last byte in the packet and always has the value of 0xA9.
- 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.
- Lux is the lux value which is a 4-byte real number represented with IEEE 754 single precision floating-point standard.
- Location is the location code as described in Table 1 and is a 1-byte integer.
- 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, location, lux]

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

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

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

Where

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

Depending of the input and output file format, the program should decide to convert data in the input file to a new format in the output file.
The users must provide both the two filenames, otherwise it is an invalid statement.

### *3.1 Case 1:*

If the input file is a data file which must have the same format as the one described in task 2.2, the output file should be a text file with the extension of ".dat" which contain the respective data packet as in Table 2.
For example:

```
C:\\lux_comm lux_condition.csv hex_packet_ee3491.dat
```

The programm should:

- Read each line of the input file, i.e.: lux_condition.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.:

  *A line of "**2,2023:11:11 01:00:00,2,400.03,bright**" exited in the file named "lux_condition.csv" is converted to*

  ***A0 0E 02 65 4E 6F A0 02 43 C8 03 D7 47 A9***
- Write each packet in one line in the output file, i.e. hex_packet_ee3491.dat
- Overide the output file hex_packet_ee3491.dat if it has been existed.
- Be able to process at least 10000 data points which means that the input file lux_condition.csv may consist of at least 10000 data lines.

### *3.2 Case 2:*

If the input file is a text file with the extension of ".dat" which contain the data packet as described in Table 2 and each packet is written in one line, the output file should be a csv file which must have the same format as the one described in task 2.2.
For example:

```
C:\\lux_comm hex_packet_ee3491.dat lux_condition.csv
```

The programm should:

- Read each line of the input file, i.e.: hex_packet_ee3491.dat
- Convert each data packet to proper data fields and one field is separated from the other by a comma as the file in task 2.2, e.g.:

  *A line of "**A0 0E 02 65 4E 6F A0 02 43 C8 03 D7 47 A9**" exited in the file named "hex_packet_ee3491.dat" is converted into*

  ***2,2023:11:11 01:00:00,2,400.03,bright***
- Write each converted data in one line in the output file, i.e. lux_condition.csv
- Overide the output file lux_condition.csv if it has been existed.

- Be able to process at least 10000 data ponts which means that the input file hex_packet_ee3491.dat may consist of at least 10000 lines.

## II. Other technical requirements:

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 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".
   - "lux_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 or location.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.: one or both the two filenames are 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, 2023:11:11 00:00:00, 50.1"
     - Time is blank or invalid, e.g.: "1,2023:11:11 00:00:, 50.1"
     - Lux value is blank, e.g.: "1,2023:11:11 00:00:00, "

     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.
   - If a sensor in data file is not included in the location file or the line including the location information of that sensor in location file has wrong data, e.g. data file has 9 sensors with ids from 1 to 9 but the location file has only 7 sensor ids 1, 2, 5, 6, 7, 8, 9 or location code in `location.csv` is blank, e.g.: "3,". The error message can be "Error 05: unknown location of sensor ID" where ID is the id of the sensor which does not presented in location file.

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 overriden. Error message can be: "Error 07: 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, 50.1"
    - Time is blank or invalid, e.g. "1,2023:11:11 00:00:, 50.1"
    - Lux value is blank, e.g. "1,2023:11:11 00:00:00, "

  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 "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.
- The input file is a text file (.dat) and it contains wrong data packet format or wrong value:
    - Wrong or missing start and/or stop bytes
    - Length byte and the packet length do not match
    - Wrong checksum byte (checksum error)
    - Time is in the future of the moment when the command is executed.

  The error message must include the line number in the input file in which the error happens, i.e.: "Error 05: data packet error at line X" where X is the line number. The first line in the input file is 1.

  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 06: 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. 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.

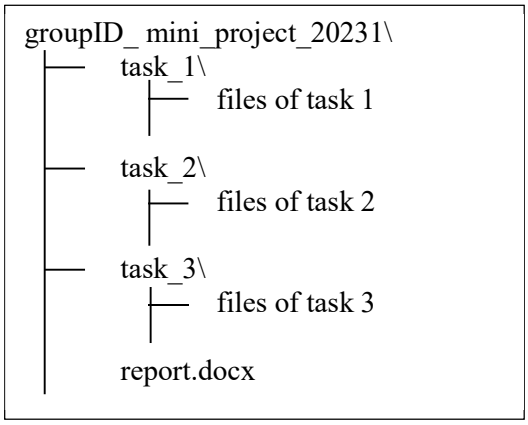
## III. **Program design:**

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 two flowcharts:

- One flowchart of the overall program (for 1 of the 3 tasks or 3 flowcharts for all 3 tasks).
- And 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_20231" and three subfolders "task_1", "task_2" and "task_3" which contain the related files. "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.

**Notice:** 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 4 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 top-down approach diagram,
    - 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),
        - 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;
        - at least two 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_20231.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.: "20221234_20222345_mini_project_20231.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 Secion III (-5%)
- The students must do the project themselves. Do **NOT** copy others' works. 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 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 ----------