Lab: a history file for the plant

1 Objective

Continue the project of *Lab strings*. Until now, the history of the volumes was kept in dynamic memory: when the program ended, it was lost. In this lab the program saves every reading cycle in a **text file** with the CSV format, which a person can open with a spreadsheet, and computes the statistics by **reading the file back**. At the end of this lab you will have:

- created a text file with `fopen()`, written it with `fprintf()` and closed it with `fclose()`, checking every step;
- read the file line by line with `fgets()` + `sscanf()`, with the read function as the condition of the loop;
- seen what the buffer of a stream, the working directory and the opening modes mean in practice.

> Estimated time: 60–90 minutes. Keep following the Barr-C style guide, and fix every warning as if it were an error.

2 The history file

The new module `logger` writes the file `history.csv`. The first line is a header, and then there is one line per tank and cycle, with the fields separated by `;`:

```
cycle;tank;raw;level_m;volume_l
0;0;1200;0.586;460.307
0;1;2000;1.465;2589.225
...
2;2;65535;;
...
```

Field	Format	Meaning
`cycle`, `tank`	integer	Reading cycle and index of the tank
`raw`	integer	Raw reading of the sensor
`level_m`, `volume_l`	3 decimals	Level in metres and volume in litres; **empty** if the sensor is faulty

In cycle 2, the sensor of tank 2 is faulty: its line keeps the cycle, the tank and the raw reading (65535), and leaves the level and the volume empty.

> CSV means *comma-separated values*, but with French or Spanish settings the comma is the decimal separator, and spreadsheets expect `;` between fields. That is why this file uses `;`.

3 Requirements

- *Lab strings* finished: the folder `lab_strings`, which builds without warnings and prints the expected output of that lab. If your version does not work, ask your instructor for the reference version.
- The file `lab_files_given.zip`, given by your instructor.
- A spreadsheet (LibreOffice Calc or Excel), only to look at the file.

4 Starting point

1. Copy the whole *Lab strings* project into a new folder named `lab_files`:

```
C:\...> cd %USERPROFILE%\c_projects
C:\...> xcopy lab_strings lab_files /E /I
C:\...> cd lab_files
C:\...\lab_files> mingw32-make clean
C:\...\lab_files> del answers.txt
```

2. Extract `lab_files_given.zip` into the project folder, replacing `src\main.c`. It contains:
   - `include\logger.h` and `src\logger.c`: the module you will complete. `logger.c` has the four functions with an empty body that always fails.
   - `src\main.c` (V4.0): the new main program. It reads the tanks through the bus as in *Lab strings*, but it no longer runs the bus and parser tests, and it no longer uses `malloc()`. It creates the history file before the cycles, adds the lines of each cycle with `logger_write()`, closes the file and, at the end, reads the volumes of each tank back with `logger_read_volumes()` to compute the statistics.
3. Build and run with `mingw32-make run`, and type `6`. The program prints the configuration, asks for the number of cycles and stops with `Cannot create the history file`: the module `logger` is still empty. Read `main.c` before you go on: find the four calls to the `logger` functions and what `main()` does when each of them fails.

5 Part 1: write the file

Complete the three functions that write the file, in `src/logger.c`.

1. Write `logger_create()`:

```
FILE *logger_create(const char *p_name);
```

   It opens the file for writing with `fopen()` (if the file exists, it is overwritten) and writes the

header line, `LOGGER_HEADER`, with `fputs()`. It returns the open file, or `NULL` if `fopen()` or `fputs()` fails. In both cases it prints the reason with `perror(p_name)`, and if `fputs()` fails it closes the file before returning. Use the pattern of the slides:

```
FILE *p_file = fopen(p_name, "w");

if (NULL == p_file)
{
    perror(p_name);
    return NULL;
}
```

`fputs()` returns `EOF` on error. `perror()` prints its argument, a colon and the reason of the last error, in `stderr`.

2. Write `logger_write()`:

```
int logger_write(FILE *p_file, uint8_t cycle,
                 const tank_state_t *p_state, uint8_t num_tanks);
```

It writes one line per tank with `fprintf()`: if `p_state[tank].sensor_ok` is true, the five fields; otherwise, only the first three, followed by `;;`. It checks the value returned by each `fprintf()`: if it is negative, the function stops and returns -1. Otherwise it returns 0. The format of a valid line is:

```
"%" PRIu8 ";%" PRIu8 ";%" PRIu16 ";%.3f;%.3f\n"
```

`cycle` and `tank` are `uint8_t`, and `raw` is `uint16_t`: that is why the format uses the `<inttypes.h>` macros, as in *Lab strings*.

3. Write `logger_close()`:

```
int logger_close(FILE *p_file);
```

It closes the file with `fclose()`. If `fclose()` fails, it prints the reason with `perror("logger_close")` and returns -1; otherwise it returns 0.

4. Build and run, and type `6`. The cycles must be identical to those of *Lab strings*, and then the program must show:

```
History saved in history.csv

Volume history (L), valid readings read from history.csv
  tank  samples        min        max       mean
     0  cannot read the history file
     1  cannot read the history file
     2  cannot read the history file
     3  cannot read the history file
```

The file is written, but `logger_read_volumes()` is still empty: that is Part 2.

5. Open `history.csv` in VS Code. It must have 25 lines: the header and 24 lines of data. Check the first lines and the line of cycle 2, tank 2 against the section *The history file*.
6. Open `history.csv` with LibreOffice Calc or Excel, and keep what you see for question 1. Close the spreadsheet before you go on: while it is open, Excel may not let your program rewrite the file.
7. **A folder that does not exist.** In `main.c`, replace temporarily `"history.csv"` with `"data/history.csv"` in the definition of `HISTORY_FILE`, build and run. Keep what the program prints for question 4, then restore the name.

6 Part 2: read the file back

Complete `logger_read_volumes()`:

```
int logger_read_volumes(const char *p_name, uint8_t tank,
                        double *p_volumes, uint8_t max_volumes,
                        uint8_t *p_num_volumes);
```

It opens the file for reading, skips the header and reads every line. For each valid line of the tank `tank`, it stores the volume in the array `p_volumes`. It returns 0 on success, and -1 if the file cannot be opened or read, or if there are more volumes than `max_volumes`. In every case it closes the file and writes in `*p_num_volumes` the number of volumes stored.

1. Open the file with `"r"` and check it with `perror()`, as in Part 1.
2. Read the header with one `fgets()`. Then read the rest of the lines with the loop pattern of the slides: **the read function is the loop condition**. Analyse each line with **one call to** `sscanf()` and the format:

```
"%" SCNu8 ";%" SCNu8 ";%" SCNu16 ";%lf;%lf"
```

A line is valid only if `sscanf()` assigned the **5 fields**, and it belongs to the tank if its second field is equal to `tank`.

For the line of a faulty sensor, `2;2;65535;;`, `sscanf()` reads the first three fields, then finds `;` where it expects a number, stops and returns 3. You do not need any special code for the faulty sensors: checking the 5 is enough.

3. After the loop, call `ferror()`: if it is not 0, the loop stopped because of a read error, not because of the end of the file. Then close the file. The structure is the one of the slides:

```
while (NULL != fgets(line, sizeof(line), p_file))
{
    /* analyse the line */
}

if (0 != ferror(p_file))
{
    perror(p_name);    /* read error */
}
```

4. Build and run, and type `6`. The statistics must be **identical** to those of *Lab strings*: now they come from the file.

```
Volume history (L), valid readings read from history.csv
  tank  samples        min        max       mean
     0        6      460.3     1457.6     1010.1
     1        6      388.4     2589.2     1251.5
     2        5      376.9      377.5      377.2
     3        6     6712.8     7575.9     7080.4
```

5. **Forget `fclose()`.** In `main.c`, comment out temporarily the whole `if` block that calls `logger_close()`, build and run with 6 cycles. Keep the statistics for question 3. Then open `history.csv` in VS Code and check whether it is complete. Restore the code afterwards.
6. **Two runs.** Run the program twice: first type `6`, then `3`. Keep the statistics of the second run. Then, in `logger_create()`, replace temporarily the mode `"w"` with `"a"`, build, and run the program twice, typing `6` both times. Keep the statistics of the second run and look at the file, for question 5. Restore `"w"` afterwards.

7 Part 3: questions

Answer these questions in a plain text file named `answers.txt`, in the project folder. Short answers are fine, but justify them.

1. How does the line of the faulty sensor look in VS Code, and in the spreadsheet? Which value does `sscanf()` return for that line? Did the spreadsheet show the levels and the volumes as numbers? What could go wrong with the number `0.586` in a spreadsheet with French

settings?

2. In the terminal, run `dir history.csv`. What is its size in bytes? On Linux, the same program writes a file of 582 bytes. Explain the difference. (Hint: count the lines, and remember the text mode of Windows.)
3. In Part 2, step 5, what did the statistics show, and why? When you opened the file afterwards, was it complete? Which part of the system wrote the last lines, and when? What could happen to them if the program crashed before the end?
4. In Part 1, step 7, what did the program print, and which line was printed by which function? Why does `main()` stop instead of going on with the cycles? What would happen if `logger_create()` did not check the value returned by `fopen()`? In Java, `new FileWriter("data/history.csv")` throws an `IOException`: what is the equivalent in C?
5. In Part 2, step 6, what did the second run show with the mode `"w"`, and why? With the mode `"a"`, how many header lines does the file have, and why did the statistics fail? Is it a good idea to open a history file in mode `"a"`? What would you have to change to make it work?
6. Open a terminal in the `build` folder (`cd build`) and run `app.exe`, typing `6`. Where is `history.csv` now? What happens to it with `mingw32-make clean`? Explain why the file is not always in the same folder.

8 Going further (optional)

1. **A data logger that survives a crash.** At the end of `logger_write()`, call `fflush(p_file)` and check its return value (0 on success). Repeat the experiment of Part 2, step 5: what do the statistics show now, and why? What is the cost of calling `fflush()` after every cycle, on a PC and on a microcontroller that writes to an SD card?
2. **Binary history with random access.** Add three functions to `logger`:

```
FILE *logger_create_binary(const char *p_name);
int   logger_write_binary(FILE *p_file, const tank_state_t *p_state,
                          uint8_t num_tanks);
int   logger_read_record(const char *p_name, uint8_t cycle,
                         uint8_t tank, uint8_t num_tanks,
                         tank_state_t *p_state);
```

   - `logger_create_binary()` opens the file with `"wb"`; the file is closed with `logger_close()`.
   - `logger_write_binary()` writes the whole array of states with one `fwrite()`, and checks that all the elements were written.
   - `logger_read_record()` opens the file with `"rb"`, moves to the record of `cycle` and

`tank` with `fseek()`, reads it with one `fread()`, and copies it to `*p_state` only if the read succeeded. The position of a record is (`cycle` × `num_tanks` + `tank`) × `sizeof(tank_state_t)`.

- In `main()`, write `history.bin` in every cycle, next to `history.csv`, and at the end print the record of cycle 3, tank 1. With 6 cycles it must show:

```
Record of cycle 3, tank 1, read from history.bin
  raw 600, volume 776.8 L, status LOW
```

Compare the sizes of `history.csv` and `history.bin`: why is the binary file bigger, if binary files are supposed to be compact? What does the program print with 3 cycles, and why? Could a program compiled for another processor read `history.bin`?

9 Checklist and submission

Your lab is complete if:

- ☐ `mingw32-make clean` followed by `mingw32-make` builds the project without any warning.
- ☐ `main.c`, `logger.h`, `nudam_sim`, `sensor`, `area`, `volume`, the `Makefile` and the files in `.vscode` have not been modified.
- ☐ Every call to `fopen()`, `fputs()`, `fprintf()` and `fclose()` in `logger.c` checks its return value.
- ☐ `logger_read_volumes()` uses `fgets()` as the condition of its loop, one `sscanf()` per line, checks `ferror()` after the loop and closes the file in every case.
- ☐ `history.csv` has 25 lines, and the line of cycle 2, tank 2 is `2;2;65535;;`.
- ☐ `answers.txt` answers the six questions.
- ☐ With 6 cycles, the program prints the expected output below.

The final output of the program, typing `6`, must be:

```
Plant configuration: 4 tanks
  tank  radius(m)  height(m)  low(%)  high(%)  capacity(L)
     0       0.50       2.00      20       90       1570.8
     1       0.75       3.00      15       85       5301.4
     2       0.40       1.50      25       90        754.0
     3       1.00       2.50      10       95       7854.0
  plant capacity: 15480.2 L

Number of cycles (1-6): 6

Cycle 0
```

```
  tank    raw  level(m)  fill(%)  volume(L)  status
     0   1200     0.586     29.3      460.3  NORMAL
     1   2000     1.465     48.8     2589.2  NORMAL
     2   2048     0.750     50.0      377.1  NORMAL
     3   3500     2.137     85.5     6712.8  NORMAL
  plant volume: 10139.4 L (65.5 % of capacity), 0 sensor fault(s)

Cycle 1
  tank    raw  level(m)  fill(%)  volume(L)  status
     0   1800     0.879     44.0      690.5  NORMAL
     1   1500     1.099     36.6     1941.9  NORMAL
     2   2050     0.751     50.1      377.5  NORMAL
     3   3550     2.167     86.7     6808.7  NORMAL
  plant volume: 9818.5 L (63.4 % of capacity), 0 sensor fault(s)

Cycle 2
  tank    raw  level(m)  fill(%)  volume(L)  status
     0   2400     1.172     58.6      920.6  NORMAL
     1   1000     0.733     24.4     1294.6  NORMAL
     2  65535         -         -          -  SENSOR FAULT
     3   3600     2.198     87.9     6904.6  NORMAL
  plant volume: 9119.8 L (58.9 % of capacity), 1 sensor fault(s)

Cycle 3
  tank    raw  level(m)  fill(%)  volume(L)  status
     0   3000     1.465     73.3     1150.8  NORMAL
     1    600     0.440     14.7      776.8  LOW
     2   2047     0.750     50.0      376.9  NORMAL
     3   3700     2.259     90.4     7096.4  NORMAL
  plant volume: 9400.8 L (60.7 % of capacity), 0 sensor fault(s)

Cycle 4
  tank    raw  level(m)  fill(%)  volume(L)  status
     0   3600     1.758     87.9     1380.9  NORMAL
     1    400     0.293      9.8      517.8  LOW
     2   2049     0.751     50.0      377.3  NORMAL
     3   3850     2.350     94.0     7384.1  NORMAL
  plant volume: 9660.1 L (62.4 % of capacity), 0 sensor fault(s)

Cycle 5
  tank    raw  level(m)  fill(%)  volume(L)  status
     0   3800     1.856     92.8     1457.6  HIGH
     1    300     0.220      7.3      388.4  LOW
     2   2048     0.750     50.0      377.1  NORMAL
     3   3950     2.411     96.5     7575.9  HIGH
  plant volume: 9799.0 L (63.3 % of capacity), 0 sensor fault(s)
```

```
History saved in history.csv

Volume history (L), valid readings read from history.csv
  tank  samples        min        max       mean
     0        6      460.3     1457.6     1010.1
     1        6      388.4     2589.2     1251.5
     2        5      376.9      377.5      377.2
     3        6     6712.8     7575.9     7080.4
```

> Submit the folders `src` and `include` and the file `answers.txt`, as indicated by your instructor. Do not submit the `build` folder or `history.csv`: run `mingw32-make clean` first. Keep the project: the next labs start from it.

10 Troubleshooting

- `unknown type name 'FILE'`. `logger.h` includes `<stdio.h>`: check that `logger.c` includes `"logger.h"`.
- `expected ')' before 'PRIu8'` (or `'SCNu8'`). Include `<inttypes.h>` in `logger.c`.
- `format '%u' expects argument of type 'unsigned int *', but argument has type 'uint8_t *'`. In `sscanf()`, a `uint8_t` is read with `"%" SCNu8`, and a `uint16_t` with `"%" SCNu16`.
- `Cannot create the history file`, **and** `history.csv: Permission denied`. The file is open in Excel: close it and run again.
- **Every tank has 0 samples.** The file is read before it is closed, or the loop never runs: check `logger_close()` and the condition of your loop.
- **Tank 2 has 6 samples.** You do not check that `sscanf()` returned 5: the line of the faulty sensor is counted, with the volume left in the variable by the previous line.
- **The last line of a tank is counted twice.** Your loop condition is not the read function. Use `fgets()` as the condition, as in the slides.
- `history.csv` **is not in the project folder.** The program writes it in the working directory: run it with `mingw32-make run` from the project folder.