Lab: a plant with cylindrical tanks

1 Objective

Start the project that you will extend in the next labs: the level monitoring of a plant with several cylindrical tanks. Each tank has a level sensor, and your program converts its readings into liquid levels and volumes. At the end of this lab you will have:

- chosen the right fixed-width integer types from `stdint.h` for each value;
- defined new data types with `typedef`: an `enum` for the alarm status and two `struct` for the configuration and the state of a tank;
- stored the configuration of the plant in a `const` array of structs, and let the compiler count its elements;
- kept the state of all the tanks in an array, updated at each reading cycle;
- reused the functions of your `volume` module.

> Estimated time: 90–120 minutes. Keep following the Barr-C style guide, and fix every warning as if it were an error.

2 The plant

The plant has four vertical cylindrical tanks. Each one has a level sensor connected to a 12-bit analog-to-digital converter (ADC). The sensor does not give the level in metres: it gives a **raw reading** between 0 (empty tank) and 4095 (tank full to the top). If the sensor is not responding, the reading is `0xFFFF` (65535).

Your instructor provides the module `sensor`, which simulates the sensors. It gives six reading cycles for each tank, always the same, so your output must match the expected output exactly. Its header, `sensor.h`, contains:

```
#define SENSOR_ADC_MAX      4095U    /* Highest valid reading (12 bits).  */
#define SENSOR_FAULT_VALUE  0xFFFFU  /* Reading of a faulty sensor.       */
#define SENSOR_NUM_TANKS    4U       /* Tanks with a simulated sensor.    */
#define SENSOR_NUM_CYCLES   6U       /* Simulated reading cycles.         */

uint16_t sensor_read_level(uint8_t tank_index, uint8_t cycle);
```

The configuration of the plant is:

Tank	Radius (m)	Height (m)	Low alarm (%)	High alarm (%)
0	0.50	2.00	20	90
1	0.75	3.00	15	85
2	0.40	1.50	25	90
3	1.00	2.50	10	95

A tank is in **low alarm** when its fill percentage (level relative to its height) is below the low alarm value, and in **high alarm** when it is above the high alarm value.

3 Requirements

- *Lab make* finished: the folder `lab_make`, which builds without warnings with `mingw32-make`. If your version does not work, ask your instructor for the reference version.
- Your `volume` module must contain this function, from *Lab shapes*:

```
double volume_cylinder(double radius, double height);
```

- The files `sensor.h` and `sensor.c` provided by your instructor.
- The C file template (`template.h` and `template.c`) of the *C build* session.

4 Starting point

1. Copy the whole *Lab make* project into a new folder named `lab_plant`, and remove what you do not need any more:

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

   Delete also the screenshot of *Lab make*, if you saved it inside the project folder.
2. Copy `sensor.h` to `include` and `sensor.c` to `src`. Do not modify them.
3. Create the new module from the C file template: copy `template.h` to `include/tank.h` and `template.c` to `src/tank.c`. Fill in the header comment of both files (`@file`, `@brief`…).
4. Open the folder in VS Code. Your project must look like this (with `surface.c` and `surface.h` too, if you did the optional part of *Lab make*):

```
lab_plant
├── Makefile
├── README.md
├── .vscode
│   └── ...
├── include
│   ├── area.h
│   ├── sensor.h
│   ├── tank.h
│   └── volume.h
└── src
    ├── area.c
    ├── main.c
    ├── sensor.c
    ├── tank.c
    └── volume.c
```

> The modules `area` and `volume` stay as they are. From now on they are a library that the rest of the program uses. Only `main.c` is rewritten from scratch in this lab.

5 Part 1: the data types of a tank

Write `include/tank.h`, keeping the sections of the template. Rename the include guard to `TANK_H`, and include `<stdint.h>` and `<stdbool.h>` in the section *Includes*. The three types of this part go in the section *Exported types*.

1. Define the alarm status as an enumerated type named `tank_status_t`, with these values in this order: `TANK_LOW`, `TANK_NORMAL`, `TANK_HIGH`, `TANK_SENSOR_FAULT`.
2. Define the fixed configuration of a tank as a structure type named `tank_config_t`:
3. Define the state of a tank after a reading as a structure type named `tank_state_t`:
   Choose the type of each field with these rules:
   - Integer values: the **smallest** type of `stdint.h` that can hold every possible value.
   - Values with decimals: `double`.
   - Yes/no values: `bool`.
   - The alarm status: the type of step 1.
4. Add the prototypes of the three functions of the module, in the section *Exported functions*:

```
double       tank_capacity_l(tank_config_t config);
tank_state_t tank_update(tank_config_t config, uint16_t raw);
void         tank_print_status(tank_status_t status);
```

Notice that `tank_update()` **returns a whole structure**. In C a structure is a value, like an `int`: it is copied when passed to a function and when returned. This is different from Java, where objects are always handled through references.

> **Why `double` and not `float`?** Your PC computes with `double` as fast as with `float`, and your `volume` module already uses `double`. On a microcontroller the decision would be different. Many of them (for example an ARM Cortex-M4F) have hardware support only for `float`, so every `double` operation is computed by software, much more slowly and with more code. There you would use `float` for all these fields, write the constants with the `F` suffix (`100.0F`), and avoid calling `double` functions such as `volume_cylinder()`.

6 Part 2: the configuration of the plant

1. In `src/tank.c`, write `tank_capacity_l()`. It returns the total volume of the tank in litres (1 m³ = 1000 L), and it must call `volume_cylinder()`. Include `volume.h` after `tank.h`, and define the conversion factor in the section *Private define*. As in the template, write a comment with `@brief`, `@param` and `@retval` before every function.
2. Replace `src/main.c` with a new copy of `template.c`, and fill in its header comment. Write a new `main()` that declares the configuration of the plant as a `const` array, with the values of the table of section *The plant*:

```
const tank_config_t plant[] =
{
    { .radius_m = 0.50, .height_m = 2.00,
      .low_pct = 20U, .high_pct = 90U },
    ...
};
```

   The syntax `.field = value` is a *designated initializer* (C99). It is optional, but it makes the table much easier to read and to check.
3. Do not write the number of tanks anywhere. Let the compiler compute it from the size of the array:

```
const uint8_t num_tanks = (uint8_t) (sizeof(plant) / sizeof(plant[0]));
```

   `sizeof(plant)` is the size of the whole array in bytes, and `sizeof(plant[0])` the size of one element.

4. Print the configuration of the plant, the capacity of each tank and the total capacity of the plant. Build and run with `mingw32-make run`. The output 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
```

> To print `%` with `printf()`, write `%%`. Use this format for the rows: `"  %4u  %9.2f  %9.2f  %6u  %7u  %11.1f\n"`. The format `%u` expects an `unsigned int`, so convert your integer types explicitly: `(unsigned int) plant[tank].low_pct`.

7 Part 3: reading cycles

1. In `src/tank.c`, write `tank_update()`. It creates a local `tank_state_t`, fills all its fields and returns it:
   - If `raw` is greater than `SENSOR_ADC_MAX`, the reading is not valid. Set `sensor_ok` to `false`, the status to `TANK_SENSOR_FAULT`, and the level, fill percentage and volume to 0.
   - Otherwise, compute the level. The reading is proportional to the level: 0 means empty and `SENSOR_ADC_MAX` means full to the top (`height_m`). Then compute the fill percentage and the volume, using `volume_cylinder()`. Finally, compare the fill percentage with `low_pct` and `high_pct` to set the status.

   Include `sensor.h` to use `SENSOR_ADC_MAX`.
2. Write `tank_print_status()`. It prints the name of the status (`LOW`, `NORMAL`, `HIGH` or `SENSOR FAULT`) without a new line, using a `switch`. Add a `default` case that prints `UNKNOWN`.
3. In `main()`, declare the array that keeps the state of every tank:

```
tank_state_t state[sizeof(plant) / sizeof(plant[0])];
```

4. Add a loop over the cycles, from 0 to `SENSOR_NUM_CYCLES - 1`. In each cycle, for each tank:
   - read its sensor with `sensor_read_level()`;
   - update its state with `tank_update()` and store the result in `state`;
   - print one row with the reading, the level, the fill percentage, the volume and the status. If

the sensor is faulty, print `-` instead of the three numbers.

At the end of each cycle, print the total volume of the plant (tanks with a faulty sensor are not counted), its percentage of the plant capacity, and the number of faulty sensors. The output of the first cycles must be:

```
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
...
```

The complete output is in section *Checklist and submission*.

Formats for the rows: `"  %4u  %5u  %8.3f  %7.1f  %9.1f  "` for a valid reading, and `"  %4u  %5u  %8s  %7s  %9s  "` with the strings `"-"` for a faulty one. Then call `tank_print_status()` and print the new line.

5. **A classic mistake.** In `tank_update()`, replace your formula for the level, temporarily, with this one:

```
state.level_m = (raw / SENSOR_ADC_MAX) * config.height_m;
```

Build and run. Note what happens to the levels, the volumes and the alarms, and whether gcc reported anything. Keep the answer for question 2, then restore your formula.

6. **Add a tank.** Add a fifth tank to `plant`: radius 0.60 m, height 2.00 m, alarms 20 % and 90 %. Build and run without changing anything else. Keep the result for question 3, then remove the fifth tank.

8 Part 4: questions

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

1. Which types did you choose for `raw`, `low_pct` and `high_pct`, and for the loop variables of the tanks and of the cycles? Justify each choice with the range of values it must hold. Why is the type of `low_pct` not big enough for `raw`?
2. In Part 3, step 5, what are the values of `raw / SENSOR_ADC_MAX` when `raw` is 2048, and when it is 4095? Why? Why did gcc not warn you? Write the formula that you finally used and explain why it works.
3. In Part 3, step 6, which lines of `main()` did you have to change so that the program handled

five tanks? Why was the fifth tank reported as `SENSOR FAULT` in every cycle? What advantage does `sizeof(plant) / sizeof(plant[0])` have over writing `#define NUM_TANKS 4`?

4. What integer value does each constant of `tank_status_t` have? Add the line `state[0].status = 7;` at the end of `main()` and build: does gcc complain? What would `tank_print_status()` print for that value? Compare with what Java would do with an `enum`. Remove the line afterwards.
5. Add the line `plant[0].height_m = 3.0;` at the end of `main()` and build. What does gcc report, and why? Why is it a good idea to declare the configuration as `const`, while the array `state` is not `const`? Remove the line afterwards.
6. From cycle 1 to cycle 2, the plant volume drops by almost 700 L. Compare the volumes of each tank in both cycles: which part of that drop really happened, and which part did not? Do you think the total of cycle 2 is a good estimate of the real volume? Propose a better behaviour.

9 Going further (optional)

1. **Keep the last valid value.** Add to the `tank` module this function:

```
tank_state_t tank_update_hold(tank_config_t config,
                              tank_state_t previous, uint16_t raw);
```

   It works like `tank_update()`, but when the reading is not valid it keeps the level, fill percentage and volume of `previous`, which is the state of the previous cycle. The status is still `TANK_SENSOR_FAULT`. Use it in `main()` instead of `tank_update()`, and print the numbers also for faulty sensors. Then:
   - In cycle 2, tank 2 must show 0.751 m, 50.1 % and 377.5 L, with status `SENSOR FAULT`, and the plant volume must be 9497.3 L.
   - Before the first cycle there is no previous state. Make sure that every element of `state` has a valid value before the loop starts. Explain in `answers.txt` how you did it.
2. **Size of the data.** Print `sizeof(tank_config_t)`, `sizeof(tank_state_t)`, `sizeof(plant)` and `sizeof(state)`. Add up the sizes of the fields of each structure by hand and compare. Are the results equal? Search for *structure padding* and explain the difference in `answers.txt`. On a microcontroller, which of these arrays could be stored in flash memory, and which must be in RAM?

10 Checklist and submission

Your lab is complete if:

- ☐ `mingw32-make clean` followed by `mingw32-make` builds the project without any warning.
- ☐ `sensor.h`, `sensor.c`, the `Makefile` and the files in `.vscode` have not been modified.
- ☐ `tank.h`, `tank.c` and `main.c` follow the C file template, with a comment before every function.
- ☐ The number of tanks is not written anywhere in your code: it is computed with `sizeof`.
- ☐ `tank_capacity_l()` and `tank_update()` call `volume_cylinder()`.
- ☐ `answers.txt` answers the six questions.
- ☐ The program prints the expected output below.

The final output of the program 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

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)
```

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

11 Troubleshooting

- `unknown type name 'uint8_t'` **or** `'bool'`**.** A file uses these types without including `<stdint.h>` or `<stdbool.h>`. Include them in `tank.h`, and in every `.c` file that uses them.
- `unknown type name 'tank_config_t'`**.** The file does not include `tank.h`, or includes it after the code that uses the type.
- `undefined reference to 'volume_cylinder'`**.** `volume.c` is not in `src`, or the function has a different name in your *Lab shapes* module.
- `excess elements in struct initializer`**.** A row of `plant` has more values than the structure has fields, or a closing brace is missing.
- **All the levels are 0.000 and all the tanks are** `LOW`**.** Read Part 3, step 5 again.
- **The numbers differ from the expected output in the last decimal.** Check that you use `double` everywhere and the formats of the hints.
- `multiple definition of 'main'`**.** There are two `.c` files with a `main()` function in `src`. Keep only one.