Scientific Computing for Mechanical Engineering [4EM30]
— *Instruction 1* —

Getting started with C

We will use the Pelles C development environment, which can be downloaded for free from:

`http://www.smorgasbordet.com/pellesc/`

Go to download in the menu on the left-hand side of the website and select the correct version of Pelles C. Most likely, your laptop has a 64-bit architecture, which means you will need the file setup 64-bit edition. Please make sure that you download and install version 8.00. There is no need to download and install the SDK add-in, unless you plan to develop apps for mobile phones in your own time.

The Pelles C program works out of the box and contains a nice and extensive help file. You can also find a lot of documentation on the internet. The following sites are worth visiting:

`https://www.cprogramming.com/tutorial/c-tutorial.html`
`https://www.tutorialspoint.com/cprogramming/index.htm`

In order to have a good overview of your source code, it is recommended to use a fixed width font in the editor. You can change the font as follows. Go to the item "Options" in the "Tools" menu. When you click on this, a new window pops up. Under the tab "Source", you can select the font and change its size. A nice fixed width font is "Courier" in combination with a font size of 10 pixels.

The source code for the instructions is available on GitHub. We will explore the possibilities of Git and GitHub in more detail in the upcoming lectures and instructions. For this instruction the source code can be downloaded from the website:

`https://github.com/TUE-4EM30/Instructions`

Click on the "Download ZIP" button and store and extract this zip file on your computer. All files required for this instruction can be found in the "Instruction1" folder. Note that no GitHub account is required to download this zip file. However, remind to create a personal GitHub account in response to the e-mail you received last week.

Round off errors

Open the project "example1a" by double-clicking the Pelles C project file `example1a` (the one with the beach-ball icon) in the corresponding directory. The program consists of a single file `main.c`, which is shown in the editor.

1. Take a close look at the program and see if you understand every line of the code. Without running the program, try to imagine what will be printed on the screen.

2. Compile and run the program. This can be done at once by clicking the "Execute" icon, on top of the screen (this is the icon with the white arrow pointing to the right). A new, black window opens, which shows the output of the program. Is the output in agreement with what you expected? Try to give an explanation for what you observe.

3. Open de project `example1b`. This project is identical to `example1a`, except for the fact that the variables x and d are now double precision, instead of single precision. This means that their values are stored using twice as many bits as a single precision float (i.e. twice as accurate).

4. Run the program and look at the output. What differences do you see? Can you explain this?

The problem in these examples is that a floating point (single or double precision) is stored in binary form, which causes round off errors. When in reality the value of a float should be exactly equal to 0.1, in a computer program, this is not necessarily the case. So, depending on the precision of the float and the machine architecture, a comparison of two floats may give unexpected results, as shown in this example.

Luckily, integers do not have this problem. In a C program, the value of an integer is exact, which allows you to compare it to another integer. In the next example, we will see how to use integers to avoid these problems.

Cash Machine

Cash machines are designed to pay back a certain (small) amount of money by returning the correct number of coins. An example of such a machine can be found in vending- or parking-ticket machines. The source code for the corresponding algorithm is given in `example2a`.

The user can enter an amount and the algorithm will calculate the number of coins that are needed to sum up to this amount. Note that only 1 and 2 euro coins and 5, 10, 20 and 50 euro cent coins can be used.

1. Open the project "example2a" and give it a close look. Try to imagine how the number of coins is determined that is needed to match an amount of 5 Euro and 75 eurocents.

2. Execute the program and enter 575 when asked for the amount in eurocents. Check the result. Is it correct?

3. Run the program again. Enter a requested amount of 379 eurocents. Is this result correct?

The problem here is that the algorithm is not designed to break down 3 euro 79 in the appropriate amount of coins. In this case, the limitations of the algorithm are not checked, which may (and will) lead to inaccurate results.

4. In this case, the amount that is entered should be:

    - Divisible by 5 (cents).

- Positive (the machine can only return coins)
- Less than 1000 cents (one would like to avoid cash machines with too many coins in stock).

Change the code in such a way that it aborts when the requirements above are not met.

In order to avoid that program source codes become too large and therefore unreadable by the user, it can be split in a number of smaller pieces. These pieces are called functions. For this vending machine algorithm this is done in `example2b`.

In this project, the file `main.c` only contains the major steps in the algorithm, e.g. initialise the variables, read the input, split the amount and write a report to the screen. All functions have moved to a file `mylib.c`. The declaration of the function is stated in `mylib.h`.

5. Study the new example carefully and make sure that you understand every aspect of it.

6. Run the code and test it for the amounts 575 eurocents and 379 eurocents.

7. Add a new function check that performs the same checks as mentioned in step 4. You may call this function `check`.

An advantage of C is that it allows you to create new types, which is a collection of other types, the 'structure'. In this case, you can think of a coin as a new variable, which has a value (the real value of the coin), a counter (how often it has been used), and a name. This can be implemented as follows:

```
typedef struct
{
  int  count;
  int  value;
  char *name;
} Coin;
```

Next, an array of the type Coin is created.

```
Coin coins[10];
```

And initialised in the function init:

```
coins[0].value = 200;
coins[1].value = 100;
coins[2].value = 50;
...
coins[0].name = "2 Euro";
coins[1].name = "1 Euro";
coins[2].name = "50 cent";
...
```

8. Open the project `example2c` and study it. Again, add the function check that is mentioned in step 7.

Another advantage of the use of functions in combination with library files (e.g. `mylib.c` and `mylib.h`) is that you can test specific parts of the code. Open the project `example2ctest`. It contains a different `main.c` file, but uses the mylib files of the original `example2c` project. It can be used to test specific functions in the mylib files.

9. Add tests for the functions `reset` and `check`.

Finally, the vending machine will be exported to Portugal, where 1 and 2 eurocents are still being used.

10. Extend the code (`example2c`) such that it can use 1 and 2 eurocent coins as well. Test the code for the amount of 378 eurocent. Note that because of the structure of the program and the use of functions, this requires only a little amount of work.