1 Preparation

You will need a compiler and further development utilities. On the Mac, open the App Store, and download Xcode. Then install the Xcode command line utilities. The commands below are to be entered into a terminal window, which is a program that ships with OS X. For Windows, install MinGW, including it's bin folder to the system's PATH.

2 Compiling the test program

We are using multiple C programs for this lab. You can download the source files to the current directory using the version control system *git*. Simply run the command:

```
git clone "https://github.com/johnfxgalea/Embedded-Systems-Sheet1.git".
```

Version control systems allow to manage and access different versions of a software project. We provide a more detailed introduction to the various software development tools with lab sheet 3 (on Wednesday).

Switch into the directory *ex-01* in the provided exercises archive.

```
cd ex-01
```

The folder contains a single .c file with the examples for this lab. The file needs to be compiled using a C compiler into an executable program. We use the *gcc* compiler. Compile the file to an executable program using the following command:

```
gcc -o ex-01 exercise.c
```

You can now execute the resulting program:

```
./ex-01
```

Open the C file in your preferred editor. Answer the following questions and adapt the C code as advised in the exercises.

3 Warm up: Types

One of the major features introduced by C is the type system. C manages its program data in different variable types. Each types has its own weaknesses and strengths and is used for a specific purpose.

When preparing to process data or implement a function, programmers need to decide which data type is suitable for which form of data.

Task 1 Name a suitable C data type for each of the following use cases:

- Numbers in the range $[0, 100000)$
- Floating point numbers
- English literature text
- A series of numbers in the range $(-100000, 100000)$

- Large binary data (e.g. bitmap image data)
- A boolean truth value.

Task 2 What is the purpose of the type modifier *const*?

Task 3 What does the type modifier *volatile* guarantee? When is this useful?

4 Common type issues

Take a look at the function *run_types_exercise()* in the provided code. Each of the following tasks corresponds to a code segment in this function.

Task 1 Why is the text starting with "T1" printed?

Surprisingly, the condition $-1 > 248$ seems to evaluate to *true* in the code segment. Explain why this is the case.

What do you conclude form this situation about comparing variables of different types? What do we call this effect in C?

Task 2 How is the constant "-1" printed if we use %u instead of %d?

The placeholder *%d* advises the *printf(...)* function to interpret the input variable as a signed integer. It prints the expected value *-1*. What is printed if you use *%u* instead? Explain why.

Does it matter if we pass a signed or unsigned int to *printf(...)*? Which C mechanism is used to allow for its varying parameters?

Task 3 Why does the *for* loop not terminate properly?

The expectation is that the implemented loop would count down from *10.0* to *0.0* in steps of size *0.1*. However, the loop continues beyond this limit and is only terminated by the *break* statement. Explain this behaviour. Draw conclusions from this effect and explain how this affects how you would use certain types in C.

Task 4 Inspect the illustrated up- and downcasts.

C casts do more than just interpret a binary pattern in a different fashion. Casts can drastically change the binary values they are applied to. Section "T4" in the code illustrates this.

What is the decimal value of the variable *initial*? What is the value of *down_cast*? Explain why.

Finally, what is the value of *up_cast*? What does this tell you about the *up_cast* that just took place?

In the second down- and upcast, this behaviour is not present. The values of *initial* and *up_cast* are the same. What's the difference in this scenario?

5 Cube Calculation

Functions are useful to avoid code replication, as well as improve code readability. Create a program that asks the user to input a float, and print out it's cubed value. The cube operation must be implemented in a separate function. The output value must be presented up to 2 decimal places.

6 Bitwise operations

C provides bitwise operations for its integer types. It applies the respective operator to each bit pair of two integer variables.

C provides the following bitwise operators:

&	AND
|	OR
^	XOR
~	NOT

The following listing provides an example for each operator:

```
  1011  0110        1011  0110        1011  0110
& 1001  0010      | 1001  0010      ^ 1001  0010      ~ 1001  0010
-----------       -----------       -----------       -----------
  1001  0010        1011  0110        0010  0100        0110  1101
```

Use these binary operators to implement the corresponding functions such that the test cases in *run_bitwise_exercises()* pass.

7 Coordinates exercises

Implement the function *distance_between_coordinates(...)* such that all tests in *run_coordinates_exercise()*. The goal is to calculate the distance between the two points provided as arguments.

8 Time Conversion

Build a program that asks the user to input time in minutes, and outputs its equivalent value in hours and minutes. In order to achieve this task, make use of the modulus operator (%). Once completed, extend the program such that it keeps on looping, asking for other values and printing their results, until a value of -1 is given as input.

9 Echo program

We use the standard C functions *scanf(...)*, *printf(...)*, *strcmp* to implement an echo program. The program works as a command prompts, reading text input from the console. Whenever the user enters a line terminator, i.e. presses the *RETURN* key, the text entered up to this point should be echoed. This process should quit as soon as the users enters the line *"exit"*.

10 Recursion

Build a program that prints the values from 0 to a integer value given as input using recursion. Function calls need to be used to achieve this task, without using any loops.