Note: Use the same project for exercises 1 and 2. Add three files to the project – the header file for declarations of structs, enums and typedefs and for the prototypes of the functions, a c-file for the functions and one for the test program.
Exercises which are not finished in class must be completed at home!

Learning outcome of this set of exercises: Data structures, Functions with data structure parameters, pointers to structures, arrays of structures, type definitions, enumerations.

Exercise categories:

- A – very basic, intended for inexperienced developers
- B – fair, a little bit more complex but still for starters
- C – challenging, complexity is higher, additional programming constructs may be required

Exercise 1 – Modulo n counter (A)

Write a small program which uses a data structure to store the properties of a modulo n counter, which counts from 0 to n-1, just beginning at 0 again after having reached n-1 (overflow).

Declare a data structure for counters with integer members for:

- the maximum count value (n)
- the current count value

Define a new Type for the data structure using typedef.

Write functions which

- initialize a counter by setting the current count value to 0 and the maximum count value to an arbitrary value.
- count, that is increase the current count value of a passed counter by 1 or reset it to 0, in case of the value becomes greater than n-1 (overflow). It returns 1 if an overflow had occurred otherwise 0.
- optionally print the name of the counter (modulo n counter, n should be replaced by the current value of the maximum count value) or the current count value (without newline!). Use an enumeration type to pass the option.

Test the code with a modulo 10 counter, print its name, count three times until the overflow while printing the actual count value after each counting step.

Example output:

```
modulo 10 counter
1
2
3
4
5
6
7
8
9
0
1
...
```

Exercise 2 – Multiple digit counter (B)

The program of exercise 1 has to be expanded in order to handle not only counters with one digit but with m digits. For that, use arrays of m counters.

Write additional functions which

- create an array of m counters and initialize the counters by setting their current count values to 0 and their maximum count value to one of the following values – 2 (binary), 8 (octal), 10 (decimal) or 16 (hexadecimal). The function returns ~~a pointer to~~ the newly created m digit counter or the NULL pointer in case the maximum count value is not allowed or the number of digits is 0.
  Hints: ~~You have to allocate dynamic memory! Why?~~
  Of course, each counter of the array must have the same maximum count value.
- increase the actual count value of a passed m digit counter by 1 or resets it to 0 in case of an overflow.
- optionally print the name of the counter (m digit binary/octal/decimal/hexadecimal counter, m and the type should be replaced by the current values) or the current count value of the m digit counter (with one space after the digits!). Use the ~~enumeration type~~ to pass the option.
  Note: Consider, that hexadecimal digits have values between 0 and 9 and 'A' and 'F'.
- free the memory of an array of counters.

Whenever it makes sense, call the functions of the modulo n counters from exercise 1.

Test the code. The user enters the number of digits (m) and the maximum count value of the counter (n). The program, creates a counter, prints its name and counts until the overflow occurs while printing the current count value after each counting step.
If the user enters an invalid maximum count value, the program stopps. ~~Don't forget to free the memory!~~
Try to format the output into 8 columns if n equals to 8, 10 columns if n equals to 10 and 16 columns if n equals to 2 or 16.

Example output:

```
Please enter the parameters of your counter:
number of digits: 4
type (2/8/10/16): 2

4 digit binary counter
0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
0000 0001

Please enter the parameters of your counter:
number of digits: 2
type (2/8/10/16): 16

2 digit hexadecimal counter
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
…
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
…
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
00 01
```

```
Please enter the parameters of your counter:
number of digits: 2
type (2/8/10/16): 8

2 digit octal counter
00 01 02 03 04 05 06 07
10 11 12 13 14 15 16 17
20 21 22 23 24 25 26 27
…
70 71 72 73 74 75 76 77
00 01

Please enter the parameters of your counter:
number of digits: 2
type (2/8/10/16): 10

2 digit decimal counter
00 01 02 03 04 05 06 07 08 09
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
00 01

Please enter the parameters of your counter:
number of digits: 0
type (2/8/10/16): 0

End of the test!
```