Practice 1

Mandatory

1. Tricky examples (what do these codes do?):
   http://ldani.web.elte.hu/ip/4/2-what/

2. How many ways can you print 42 to the screen? Use only signed, integer types and literals. Use only one char variable. Don't use any operation other than assignment. (Hint: octal, hexadecimal literals, type conversions, overflow) - More than 20 solutions are possible.

3. Guessing game: Read a number until it's equal to a chosen number (first it can be a hard-coded constant or #define, or you can check rand() later). In case of wrong guessing help the user by printing whether the guessed number is to small or too big. Try solving this task with different kinds of loops, and try using ?: operator instead of "if" statement.

4. Count the number of wrong guesses and give a textual evaluation in the end about this.

5. Restructure the program so there are 3 other functions besides main() (e.g. get_target, guessing, evaluate). There should be a forward declaration for all functions.

6. Try "obfuscating" your previous guessing game solution, so you use as many number literals as possible (different number systems, types other than int

- type conversions, using #define). Program semantics shouldn't change.

7. Write two functions that run alternately. One, called a, divides the value obtained as a parameter by 2 and then calls function b with the result if it is greater than 0. Function b subtracts 1 from the value obtained and then calls t with the reduced value if it is even greater as 0. Ask the user for the starting number and count how many ab iterations occur until it reaches 0.

8. Calculate what percentage power is obtained if we score 17 out of a possible 20 (whole division problem - let's discuss how good it will be and how the result will be 0).

Optional

1. Read the characters 0-9, a, b, c, d, e, f (add an error to everything else). The scanned sequence is interpreted as a hexadecimal number and then converted to a 10-digit number system. Take advantage of character-number "interoperability."

2. Write a program that decides whether a leap year is a number requested from standard input. A leap year is a year divisible by all four, except that which can be divided by a hundred. However, those divisible by four hundred will also be leap years. (DO NOT use if in the solution!)

3. Write down how many bytes your machine represents an int, unsigned, long, short, float, double, long double value (this is a duplicate from week 2 - disregard)

4. Declare a variable of type int. Calculate the maximum value that can be stored in this. Add this number to the variable, and then add one in a subsequent statement. Write the increased value. (this is a duplicate from week 2 - disregard)

5. Declare an unsigned int variable and follow the steps in the previous task.

Advanced

1. Draw the first n rows of the Pascal triangle (ask for n). Then read what bit operations C supports (and what practical benefits they can bring): https://www.geeksforgeeks.org/bitwise-operators-in-c-cpp/