WORK PACKAGE 1: C PROGRAMMING

EXERCISE 1: IF-STATEMENTS

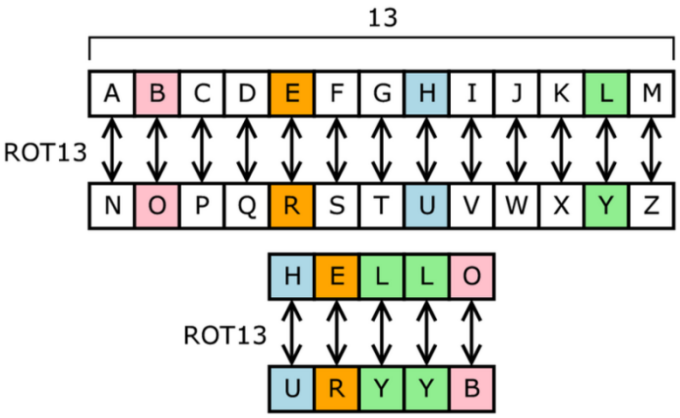
Write a program that reads in an integer number between 1 and 5 from the keyboard and prints out one of existing five sentences on the console depending on what number was entered. You can invent the sentences as you want, but each sentence should be unique.

The program continues to ask for a new number and exits if number isn´t in the interval 1 to 5.

EXERCISE 2: ENCRYPTION

Create a very simple encryption program. The program "shifts characters" in the ASCII-code table of a given input string.

In the example below, A has shifted 13 characters in the ASCII table to N, B to O, etc. The word HELLO becomes URYYB after encryption.



The user enters a text and the program prints out the encrypted text. Let the program read character by character, and encrypt it as above. The program is repeated until EOF indicated the program ends. EOF means End Of File and is a special character, e.g. when the user enters Ctrl +Z for Windows and Ctrl + d for Linux system.

The number of characters to shift should be provided as a program argument, e.g. `my_program.exe 13`

Example of a test run:

```
HELLO (+enter)
URYYB
Banana (+enter)
Onanan
 ( +Ctrl-z)
```

(Program ends)

EXERCISE 3: GUESS THE NUMBER GAME

You should develop a very simple game in which the computer creates a random integer number, between 1..100, and the user then tries to guess the number. The program should work as specified below:

- The computer creates a random number
- The user guesses the number

- The computer responds by printing one of the following:
    - You have guessed `AA` times and your guess is correct (if the user guesses correctly)
    - Your guess is too low or too high, depending on the mistake of the user
- If wrong the user is asked for a new guess, this will continue until the guess is right or the number of guesses exceeds the value `MAX_NUMBER`.
- After end of one round the user is asked for a new round or to finish.

The program should only except guessed numbers in the range of 1 …100.

An option, but not a demand, is to secure that the program does not fail (crashes) if a user by accident inputs a number outside of the range or a character/string.

EXERCISE 4: NUMBER CONVERSION

You should write two programs, which convert between numbers in different format. You should make sure that these programs can be executed in a pipeline.

The programs should accept any number between 0 and the maximum value of type `long` in C for your compiler.

The program should use the smallest possible datatype for a given, number. For example, if the user inputs 12, the program should use treat this as 8 bits and format the output accordingly. If the user inputs 1200, then the program should format the output as 16 bits, etc.

The first program must convert a number in a decimal format to a binary format.

`dec2bin.exe 12` should result in `00001100`

The second program should convert binary to hexadecimal value.

`bin2hec.exe 00001111` should result in `0F`

You should use the program arguments so that it is possible to use these two programs like this:

```
dec2bin.exe 12 | bin2hec.exe
```

The programs should be fail-safe, i.e.

- Should check if the string of the argument contains the correct digits
- Should check be able to provide help if the user provides the parameter '-h'
- Should output an error message if the conversion was not successful
- Should return 2 if the conversion is unsuccessful

EXERCISE 5: POINTERS

You should develop a program that calculates statistical values for an array of integers. Among other things, the program should plot a histogram for the frequency of different numbers in the array.

You need to create an array of integers (*table* [MAX]) where MAX is the number of random numbers between 0 and MAXNUMBER. Then you should write a function that for each possible number between 0 – MAXNUMBER calculates how many times the number exists in the array. The result is then stored in a new array (*frequency* []).

Finally, you should write a function that takes the array *frequency []* as a parameter and draws a histogram as in the example below.

Example:

Given an array `table[]={1,2,12,5,1,0,0,5,9,12,0,2,3,0}` the program will printout:

```
0   xxxx
1   xx
2   xx
3   x
5   xx
9   x
12  xx
```

Note! Numbers with frequency 0 in the array *frequency[]* are not printed out.

Please start with the skeleton code below:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100        // Defines the maximum number of the values in the table
#define MAXNUMBER 20   // Defines the maximum value of random numbers


// ------ Function declarations   ----------

// This function generates a set of random numbers
// and fills the table *tab with these numbers
void create_random(int *tab );

// This function takes the *tab of random numbers
// and creates a table with the frequency counts for these numbers
void count_frequency(int *tab, int *freq );

// This function takes the frequency count table
// and draws a histogram of the values in that frequency table
void draw_histogram(int *freq );

// ------ Function definitions   ----------


// ------ Main   -------------------------

// The main entry point for the program
//
// If you choose to go for the optional part
// Please modify it accordingly
int main (void){

    int table[MAX], n ;
    int frequency[MAXNUMBER];


}
```

Optional: instead of using `#define MAX` and `#define MAXNUMBER`, you can use the program arguments instead.