Exercise 7: Repetition, Pointers

ZOOM link:
https://unibe-ch.zoom.us/j/642646120
FAQ link:
https://docs.google.com/document/d/1crN_GsWnHgQ33gROJxEiBiVIeHfz8rpcGPeSTI7U51I/edit?usp=sharing

Before you start with these exercises, you should have done the following exercises:
3.3 (quadratic equations)
4.4 (primes)
5.5 (filter)
6.4 (weather)
6.5 (game)

7.1 Function

Rewrite the program you have written to solve quadratic equations (Ex. 3.3) into a function with the following declaration:

```
int square_equation(float a, float b, float c, float* r1, float* r2)
```

This means you give a, b, c to the function, and two pointers to floats, where the function will store the results using the addresses. The function should return the number of solutions it found (this is why it returns an int)

Test the function in a main program.

7.2 Pointer of an array

Pointers are often used when we don't know in advance how many elements an array will have. In this exercise we want to cover this issue.

Create a program in which the user can insert as many numbers as he want. I suggest you ask the user after each number: "Do you want to insert another number?" (The user can answer with 0 or 1 to make it easier for you).

You start with a certain array size (with malloc) and keeps track of the number of entries. As soon as the limit is reached, another array is created (with malloc) which has typically double the size. The data inside the old array is then copied to the new array, and the program can continue. Of course, don't forget to delete the memory allocated to the old array.

For debugging purposes, you should print out something like a log: "I see that the array is now too small. Increasing size from 8 to 16..." etc.

If you want, you can integrate your old program with the mean value and standard deviation to calculate these values after the user has inserted the numbers. Put it in a function with the array pointer as parameter and the current number of elements in the array.

You will run into many problems, Segmentation faults and other problems. The assistants will help you.

Hints

Function implementation.

```
// declaration
int square_equation(float a, float b, float c, float* r1p, float* r2p);

int main(){
    ...
    float r1, r2; // object
    // function call
    square_equation(a, b, c, &r1, &r2); // give the addresses of r1, r2
    .
    printf(...);
    ...
}

// function implementation
int square_equation(float a, float b, float c, float* r1p, float* r2p){
    ...
    *r1p = value; // assign value to an object pointed by the r1p
   pointer (in our case r1 in main function)
    ...
}
```

How to allocate/free memory.

```
// memory allocation with malloc.
// (void *) malloc( int size_in_bytes )
// You need cast into pointer type of which you want.
// sizeof() function helps to find size of data types.
int n = 10; // number of elements
float *arrayf = (float *) malloc( sizeof(float)*n );
int   *arrayi = (int *)   malloc( sizeof(int)*n );
...
// Free the memory after use. Very important to avoid memory leak!
free(arrayf);
free(arrayi);
```