Week 4

Exercise 1: Basic use of the struct

Create a `Book` struct that contains three variables: book name, year of the publication, and shelf location. The name of the book is a string (max. 50 characters) while the year of publication and the shelf location are integers. Ask the user for information on this record and print it.

Be sure to use `struct` and a constant for the length of the string in your program. It is enough to have one record in the program in which you use to save the book information, and from which it will be printed on the screen as shown in the example below. Read the name of the book by using `fgets()`, because usually the title of a book contains blanks.

Add a newline to the end of each printed line.

Example use case:

```
Enter the title of the book:
Software Engineering
Enter the year of publication of the book:
2020
Please enter a shelf for the book:
1234
The title of the book is 'Software Engineering', published in 2020
and shelf 1234.
```

Exercise 2: Addition of matrices

Write a program that asks the user for the elements of two integer matrices, prints them to the screen calculates their sum matrix, and prints the sum matrix. Implement your solution as a 2x2-matrix.

Write a ***function*** `printMatrix(int M[ROWS][COLS], char title[])` which prints a matrix `M`. The other parameter `title` is the name of the printed matrix, in this example that names are “Matrix 1”, “Matrix 2” and “Sum Matrix”. Use `#define` to define the dimensions `ROWS` and `COLS` of the matrices. Format the minimum width to four characters by using the format `%4d` when printing.

Add a newline to the end of each outputted line.

Example use case:

```
Give Matrix 1 elements:
Row 1 Col 1:
2
Row 1 Col 2:
3
Row 2 Col 1:
-1
```

```
Row 2 Col 2:
4
Matrix 1:
   2    3
  -1    4
Give Matrix 2 elements:
Row 1 Col 1:
-5
Row 1 Col 2:
3
Row 2 Col 1:
11
Row 2 Col 2:
-8
Matrix 2:
  -5    3
  11   -8
Sum matrix:
  -3    6
  10   -4
```

Exercise 3: Command line calculator with error handling.

Write a command line calculator that can add, subtract, multiply, and divide.
Use the following characters to identify calculation operations:

+	addition
-	subtraction
x	multiplication
/	division

The program receives three command line parameters, detects the numbers and the binary operation to use, and performs the calculation using a switch-case. The program must check the input and in the event of an error, it will be able to give the following error messages:

- `You did not enter input` (in case there were no command-line parameters)
- `Invalid input (`the program does not understand the input)

A string can be converted to a decimal number, for example with the `atof()` function (ascii to float), which takes the address of the string as a parameter and returns a floating-point number.

Hint: It is easiest to manage without introducing any variables in addition to the command line arguments supplied.

Assume that that name of compiled executable program is `count`.

Example use case 1:

```
./count 4 x 5
4.00 x 5.00 = 20.00
```

Example use case 2:

```
./count -4.3 + 5
-4.30 + 5.00 = 0.70
```

Example use case 3:

```
./count 9.3 / -3.14
9.30 / -3.14 = -2.96
```

Example use case 4:

```
./count 12 - -5.23423
12.00 - -5.23 = 17.23
```

Example use case 5:

```
./count
You did not enter any input.
```

Example use case 6:

```
./count 2 2
Invalid input.
```

Exercise 4: Menu-based program for maintaining user IDs

Write a menu-based program that allows you to create usernames and associated passwords, and print them.

The username and password are strings of at most 49 characters. The user data is stored in a table whose length is 10. You should define the number of max users in the beginning of the program by `#define MAX_USER 10` and use this constant in the program.

Managing records in a table requires a third item in the record, the `ID`, which is a sequential number for internal use in the program and that indicates the location (index) of the `User` in the table.

Use the `typedef` statement to define the record `User`. In addition to the main program, the program must have a menu function that prints the menu and returns the user selection as a number.

Example use case:

```
Select an option
1) Add a new User
2) Print the Users
0) Stop
1
Enter your username:
minniemouse
Enter your password:
secret
Select an option
1) Add a new User
2) Print the Users
0) Stop
1
Enter your username:
spiderman
Enter your password:
electro
Select an option
1) Add a new User
2) Print the Users
0) Stop
2
Listed Users:
Username 'minniemouse', password 'secret', ID '0'.
Username 'spiderman', password 'electro', ID '1'.
Select an option
1) Add a new User
2) Print the Users
0) Stop
0
```

Exercise 5: Recursion and Fibonacci number

Write a recursive function `Fibo(int n)` which computes the n^{th} Fibonacci number. The Fibonacci numbers `Fibo(n)` are defined recursively by

`Fibo(0) = 0, Fibo(1) = 1,`
`Fibo(n) = Fibo(n-1) + Fibo(n-2)` (for $n > 1$)

Write a program that asks the user for the number for which you are calculating the Fibonacci number, compute the Fibonacci number by your function and display the result.

Add a newline to the end of each outputted line.

Example use case:

```
Enter the number for which the Fibonacci number is calculated:
13
233
```