Exercises part 2

Ex. 10

Write a program that reads numbers from a text file and print the average of the number read. The first row of the file contains count of numbers to read from the file (excluding the first row). For example, if the first row reads 5 then you must read five numbers after the first row and calculate their average.

For example if the file contains:

```
6
12
13.5
15
16.98
2
4
```

Your program must print:

```
Average of 6 numbers is: 10.58
```

NOTE: the first line is not included in the sum!

Your program must always check the result of read operation and print an error message if read fails. For example if the count indicates that four numbers should follow but reading fails before all numbers are read then the program must print an error message that tells that all reading failed and tell how many numbers should have been read and how many were actually read. The program must still print the average value of numbers that were read.

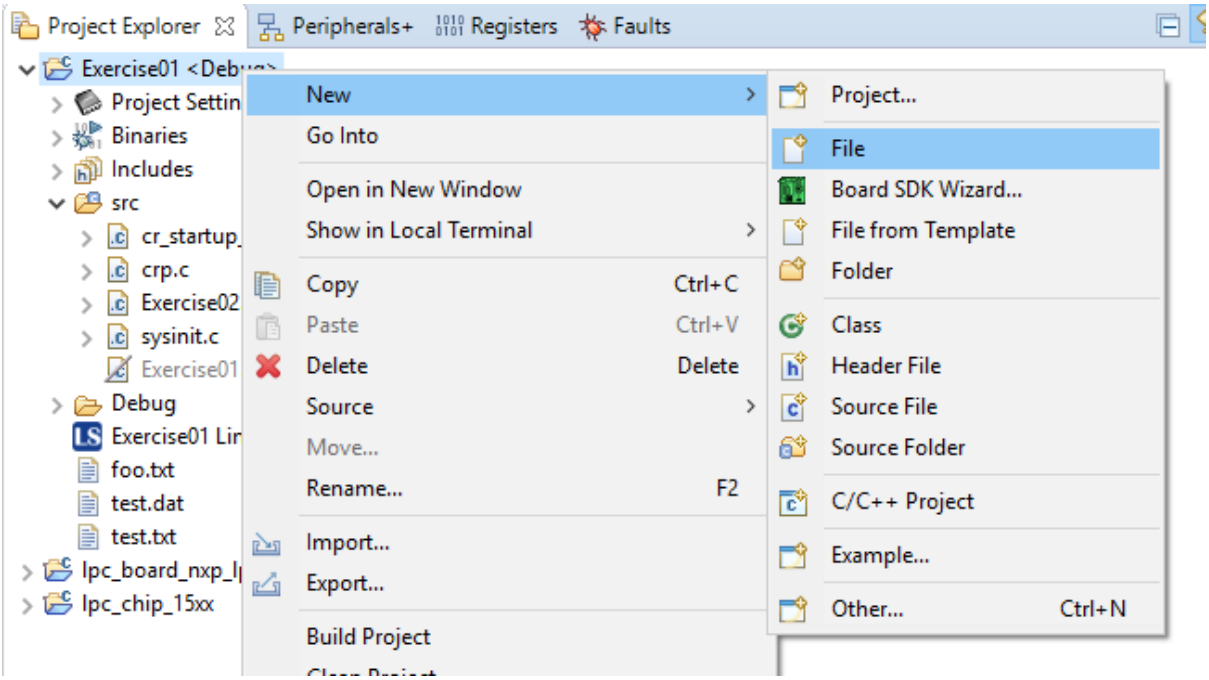
For example:

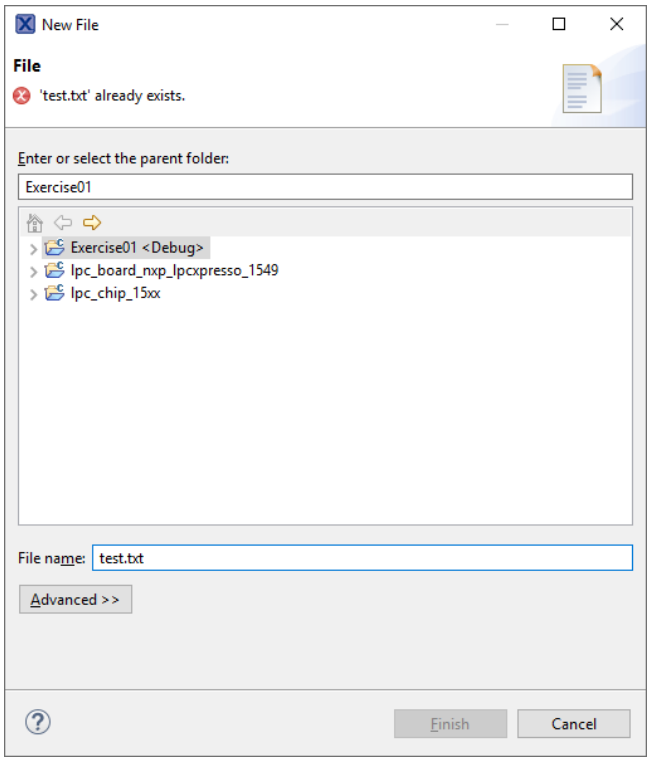
```
Error! Tried to read 6 numbers, 3 numbers were read.
Average of 3 numbers is: 3.47
```

The file to be read must be in your project directory. (See next page)

You can easily create a text file in the project directory:

1. Right click your project
2. Select New→File
3. Give your file a name and click Finish.
4. The file opens in the editor and you can edit the content





When you open a file **without a path** the file is opened in the project directory.

Ex. 11

Write a program that reads three numbers from a text file. Then the program sorts the numbers in ascending order and writes the numbers back to the file. The original file is overwritten.

Test your program with different numbers.

Ex. 12

Write a program that defines a structure to price information. The structure must contain name and price of the item. Program must define an array of structures and an integer to hold the count of items. The size of the array must be at least 20 elements and program must keep track of the actual count and ensure that array bound are not violated.

The program reads the information from a file. The name of the file is asked from the user.

Each item is stored on a single line. The price comes first and is separated from the name with a semicolon. For example:

```
9.99;USB mouse
19.98;SSD 128GB
589.90;MSI GeForce RTX 2070 SUPER VENTUS OC
59.90;HyperX Alloy Core RGB
```

The program must check that both price and name are successfully read to accept the item in the list. When the end of file is reached the program prints count of items and the items in formatted columns as shown below (Note alignment of headings and data!):

```
Name                                    Price
USB mouse                                9.99
SSD 128GB                               19.98
MSI GeForce RTX 2070 SUPER VENTUS OC   589.90
HyperX Alloy Core RGB                   59.90
```

Ex. 13

Write a program that reads strings and writes them to a file. The string must be dynamically allocated and the string can be of arbitrary length. When the string has been read it is written to the file. The length of the string must be written first then a colon (':') and then the string. The program stops when user enters a single dot ('.') on the line.

For example:

User enters: `This is a test`

Program writes to file: `14:This is a test`

Hint: `fgets()` writes a line feed at the end of the string if it fits in the string. Start with a small length, for example 16 characters, if you don't see a line feed at the end then `realloc` the string to add more space and keep on adding new data to the string until you see a line feed at the end. Then you know that you have read the whole line. Then remove any '\r' or '\n' from the string and write the string length and the string to the file. Free the string before asking for a new string.

Ex. 14

Write a program that reads a file in binary mode, divides it into 64-byte chunks and calculates a 16-bit CRC of each chunk. The program must ask user to enter the filename.

The file must be read into a dynamically allocated array of structures that contain data, data size and the CRC of the data. Note that only the last chunk may be shorter than 64 bytes. The whole file must be read before the CRC calculation takes place.

When all data has been read the program prints a list of checksums. Each line must contain chunk number, chunk size and crc. CRC must be printed as a 4-digit hexadecimal number (see printf formatting).

When the data has been printed the program must close the file and free all allocated memory.

For example:

```
struct chunk {
    uint8_t data[64];
    uint16_t size;
    uint16_t crc;
};
```

Use following code for CRC-calculation (adapted from: https://stackoverflow.com/questions/10564491/function-to-calculate-a-crc16-checksum)

```
uint16_t crc16(const uint8_t* data_p, unsigned int length) {
    uint8_t x;
    uint16_t crc = 0xFFFF;

    while (length--){
        x = crc >> 8 ^ *data_p++;
        x ^= x>>4;
        crc = (crc << 8) ^ ((uint16_t)(x << 12)) ^ ((uint16_t)(x <<5)) ^ ((uint16_t)x);
    }
    return crc;
}
```

Ex. 15

As Ex. 14 with following changes:

- After user has entered file name the program asks user to enter chunk size
- The data in the chunk structure is also dynamically allocated. The maximum size of the chunk is saved in the structure.
- In the listing of CRCs the program prints both capacity (max chunk size) and actual size of the chunk
- When listing has been printed user is asked if another file is to be read. If the answer is yes the program asks file name and chunk size again. The dynamic chunk array must be resized to according to the new parameters and the size of the file.

```
struct chunk {
    uint8_t *data;
    uint16_t size;
    uint16_t capacity;
    uint16_t crc;
};
```

Ex. 16

Write a program that allows user to remove bytes from a binary file.

Program asks user to enter file name and then displays the file size and asks user what to remove. User enters offset from the beginning of the file and number of bytes to remove. Program then overwrites the original file with the data where the user specified bytes are removed. Program must display error message if user tries to remove data past the end of the file or if the offset is invalid.

Ex. 17

In this exercise the function interfaces aren't fully specified so you need to design the interface yourself. **Note that all the information that the functions need must be passed as parameters.**

Define a structure to store car information. The structure must contain the following information: Make of the car (string), model of the car(string), price(int), co2 emissions g/km (float).

Write a program that stores the structures in a file in binary mode. You can use a fixed file name.

The program allows user to do the following:

1. print all cars in the file
2. add new car to the end of the file (ask user to enter information)
3. quit the program.

Ex. 18

Improve ex17 by adding an option where user can add new cars from a text file. The program asks user to enter the name of the file to read car information from. The file is of the following format (JSON).

```
{
”make”:”Ford”,
”model”:”Focus”,
”price”:23500,
”emissions”:187
}
```

Your program must accept the values in any order and must display an error message if data is incomplete.

Hint: read the whole file into a single string. Use strstr() to locate tags and sscanf to read the values.

Note: you don’t need to implement a generic JSON parser. It is enough to be able to read the four values described above.