SWENT 2024SS

Cross-Exercise 6

Tips:

- *Solve as many exercises as possible, at least 3.*
- *You can hand in the solutions until 23:55 the day before the next class.*
- *Only mark exercises that you can explan and show live. If you can't explain your solution sufficiently you don't get any points even for the correct solution.*
-

Exercise 1)

Write a program that defines a float array of size 5*5.
In the first row (1st subarray), write any number in the first place (read it from the console).
In each other field, write the result of the calculation of:
(array[0][0] + Index of the row) ^ (Index of the column + 1)
Then print the array using loops in the format 5 rows * 5 columns. An example where the number 3 was entered:

```
3.000000,9.000000,27.000000,81.000000,243.000000,
4.000000,16.000000,64.000000,256.000000,1024.000000,
5.000000,25.000000,125.000000,625.000000,3125.000000,
6.000000,36.000000,216.000000,1296.000000,7776.000000,
7.000000,49.000000,343.000000,2401.000000,16807.000000
```

Exercise 2)

Write a program that reads in the three vertices of a triangle in three dimensions and then calculates and outputs the perimeter of this triangle.
One vertex has three coordinates and can be entered as three floating-point numbers. Use a two-dimensional array (array[][]) that can store 3 points with 3 floating-point coordinates each.
After reading, first output the 3 x 3 coordinates (= 9 floating-point numbers) that were previously read as vertices of the triangle in the form point 0: { 1.6 4.1 -5.2 }.
Then write a function that can calculate the distance between two points in three dimensions. This function should be able to accept two points as floating-point arrays of length 3 as parameters and return the distance between the two points as a floating-point value as a return value.

Use this distance function to then calculate the three side lengths of the triangle that was read in main() and then the perimeter of the triangle. Print the perimeter at the end.
Do not forget to correctly include the math.h library for the functions sqrt() and pow().
Important: The distance function should only calculate values, return a return value, but not output anything to the console itself!
An example:

```
Enter the  x, y and z coordinates 3 times with spaces inbetween the numbers:
      0.0 0.0 0.0
      1.0 0.0 0.0
      1.0 0.0 1.0
      point 0: { 0.000000, 0.000000, 0.000000 }
      point 1: { 1.000000, 0.000000, 0.000000 }
      point 2: { 1.000000, 0.000000, 1.000000 }
      Circumference of the triangle: 3.414214
```

Exercise 3)

Write a program that defines a two-dimensional array with 20*5 entries of type int and another one-dimensional array of length 20.
Fill the two-dimensional array with random values (Attention: When using the rand() function, remember to set a seed with srand()). Write a function that sums up the random numbers in each row of the two-dimensional array and writes the result to the one-dimensional array at the appropriate index. Keep in mind that the sum of two integers can be very large and therefore use a suitable data type for the one-dimensional array. This array sum function should take as its first parameter a sub-array of the two-dimensional array, as its second parameter the length of the sub-array, and as its third parameter a pointer to the element of the one-dimensional array in which the sum should be written. Now calculate the sum for each row of the two-dimensional array with your array sum function in main(). Finally, print the 2D array as well as the 1D array with the row index in main().
A shortened example:

```
array_2D[2][5] =
          { { 4, 7, 1, 3 },   // Sum is 15
            { 6, 4, 9, 1 } }; // Sum is 20
      array_1D[2] = { 15, 20 }; // after the execution of the sum function
```

Exercise 4)

Write a program that defines a struct with the entries Id, Name, and Price. Create a 2-Dimensional Array of size 2*50 of the previously defined struct. In the first step, entries should be read from the console into the first subarray (array[0][]) in a function (with a termination condition, in order not to have to fill all entries). Return the number of entries read back to main(). In the next step, the entries should be sorted in ascending order by ID (i.e., e.g.: 1,2,3,5,6,7, etc.) and written to the second subarray (array[1][]). Finally, the user should be asked whether the entries should be output unsorted (i.e., as they were read in) or sorted by ID. Output the entries within an additional function. Note: To copy two strings (char arrays), you can use the function strncpy(), which can be found in the string.h library.

Exercise 5)

Write a program that opens two files for reading and a third file for writing, both of which contain positive, ascendingly sorted integer numbers.
The goal is to merge the two number sequences from the input files so that all the numbers from the input files are contained in the output file in ascending order.
The input files should contain ascendingly sorted positive integers, each with only one number per line.
To do this, read one number from each of the two input files, compare them, and write the smaller of the two numbers to the third output file.
Use the fscanf() function for reading (check the documentation of the function to see what happens when no more numbers can be read).
Check each time whether a number could be read from the files, as it is possible that one input file contains more numbers than the other.
Then, simply write the remaining numbers from the longer input file to the output file, as there are no more numbers to compare.
Once no more numbers can be read from both input files, close the files and end the program. Note: Text files in Linux should contain a trailing newline to be able to be read correctly.
An example:

Input File 1:	Input File 2:	Output File:
1	4	1
2	9	2
5	11	4
6	99	5
7	*Empty line*	6
Empty line		7
		9
		11
		99

Exercise 6)

Write a program that records entries in a diary-like manner and automatically appends a timestamp before each entered line.
When you start the program, all existing lines from the diary file (e.g., with the name "tagebuch.txt") should be read and displayed on the console. If there are no entries yet, the file should be created and nothing can be displayed at that point.
After displaying the diary content, you should be able to simply enter any number of lines on the console. These lines should be inserted into the diary after the existing lines, with the current timestamp inserted before each line. You can get the current timestamp by:

```
char time_buffer[100];
time_t now = time(0);
strftime(time_buffer, 100, "%Y-%m-%d %H:%M:%S", localtime(&now));
```

The character array time_buffer will then contain the system time formatted in a readable way, and you can use it like any other string.
An example:
Content of the file "diary.txt":

```
2021-06-09 15:25:21: hello
2021-06-09 15:25:25: how are you?
```

Your program should first display the following content on the command line.
then, text lines should be able to be read from the standard input.
After entering the following lines and then pressing "Ctrl+D", these additional lines should also be written to the file:

```
This is my second entry
I am well!
<STRG+D>
```

The following content should then be in "tagebuch.txt" after that (you should also be able to see this if you run your program again now):

```
2021-06-09 15:25:21: hello
2021-06-09 15:25:25: how are you?
2021-06-09 15:31:10: this is my second entry
2021-06-09 15:31:10: I am well!
```

Note: Use the fgets() function to read lines from streams. This function returns the value NULL when it reads the end-of-file indicator (EOF), which you can test (see lecture slides). You send EOF interactively by pressing Ctrl+D, so you can stop reading lines.