BI Programming. Exam

20 June 2020

General requirements

Format

- The exam will last for 120 minutes + 10 minutes for making the submission.
- Don't forget to start you SafeExamApp session!
- You can choose either Python or C# to complete the exam tasks. It is also possible to complete one task in Python, the other one in C#.
- You are free to choose any IDE
- At the end of the exam, you first need to stop the SafeExamApp session, close the application, add the session log file and your code solutions to a single ZIP archive and upload it on Canvas.

The following is strictly prohibited:

- Using electronic devices, other than your main PC
- Using headphones (wired or wireless). You can use them at the beginning of the exam to listen to initial announcements from teachers, however, after the exam time starts, you need to remove them. All communication during the exam will happen through the Zoom chat
- Communicating with other people during the exam session

Common code requirements for both tasks

- Proper code style
- Informative interaction with the user
- Structured code: using functions and possibly modules
- OOP usage is possible but is not required

Task 1

In the first task, you need to select your variant according to your Canvas ID. Use the following formula:

$$variant = CanvasID\%5 + 1$$

In this task, you don't need to implement any protection against incorrect input format. For example, if the task states that integers are requested, you can assume that the user will enter integer numbers

Task 2

In the second task, you need to implement all the necessary protection, covering cases of incorrect user input, file absense, wrong file format and others. To get maximal points for the second task, your application should not crash in any obvious case.

Scroll down for the exam tasks

Task 1

Variant 1

The program first requests a string that contains a number of time stamps in the 24-hour HH:MM format separated by a comma.

Then the program requests a positive integer number delta. Your algorithm has to shift all the timestamps by delta minutes (checking possible 24h overlap, the resulting timestamps should each be from 00:00 to 23:59) and print the modified timestamps in a single string in the same format as in the initial input (all timestamps separated by a comma)

Hint: to make calculations easier, convert each timestamp to minutes elapsed from 00:00, i.e. 05:30 = 5*60+30 = 330

Example

```
Source: 05:30, 15:45, 23:30
Delta: 45

Result: 06:15, 16:30, 00:15
```

Variant 2

Input three integer numbers: n, a, b. Generate a random nxn matrix of integer numbers, each number is in the range [a..b] (both borders included).

Then request a single string **s** having n symbols, which are all digits. You can assume correct user input here - no need to make any additional checks.

For each matrix row i, print the sum of its elements starting from index s_i, where s_i is the i-th character of s. If s_i points outside of the matrix row, print 0 as the sum.

Example

```
Matrix:
4 6 1
5 1 2
2 4 5

String: 152
Result:
7 (6+1)
0 (index 5 is outside of the row length)
5
```

Variant 3

The user enters three integer numbers: n, a and b. The program generates a random square nxn matrix of integer numbers, each number is in the range [a..b]. Find the sum of maximal values among columns and among rows

Example

```
1 4 8 3 = 8
0 8 5 9 = 9
3 4 7 5 = 7
2 1 4 9 = 9
| | | |
3 8 8 9   answer: 61
```

Variant 4

Request two non-empty strings from the user: s1 and s2. Find, how many times s2 occurs in s1. You need to solve this task doing a character-by-character analysis, avoiding creation of unnecessary additional strings (for example, these appear when using slices).

Example

```
s1: aabab
s2: ab

Result: 2
```

Variant 5

Input positive integer numbers N (number of rows) and M (number of columns) from the keyboard. Generate a random NxM matrix of latin lower case characters (a-z). Print the matrix to the console. Transpose the matrix and print it to the console. Both matrices have to be printed in a tabular format.

Remark: transposal should result in a new matrix created, it should not be an output of the same source matrix in a different direction.

Example

```
N = 2
M = 3
Source matrix:
a h x
e t z

Transposed matrix:
a e
h t
x z
```

Task 2 - same for all students

The CSV file that is given to you, contains a set of records of mobile phone calls. Each line in the file specifies the following items:

- Caller phone number
- Recipient phone number
- Date and time of call
- Duration of call in seconds. If this duration is non-negative, the call was accepted by the recipient, -1 indicates a rejected call, -2 indicates a missed call

The initial dataset has no sorting order. Changing the CSV file in an external application before using it in your program is not allowed.

Implement the following logic of the program. At startup, load the entire CSV dataset to memory, then request a phone number from the user (request repeatedly until the user either enters an empty string, in which case the program should terminate immediately, or a phone number that exists in the call database). For clarity of the following explanation, this phone number will be called source (S). After inputting the source phone number, proceed to the following main menu:

1. Call statistics: when selected, the program should print the following information in any suitable format:
    - Number and average time of accepted outgoing calls
    - Number and average time of accepted incoming calls
    - Number of missed incoming calls
    - Number of rejected incoming calls
2. Call history with other contact: the user enters another phone number (will be referred to as destination - D), the program prints all the history of calls between S and D sorted in descending order of date and time (starting from the most recent calls). The same information should also be written to a text file named S-D.txt, where S and D are the two phone numbers under consideration. The call information should be shown in relation to S. That is, if S phoned D, this should be marked as outgoing call, if D phoned S, it should be marked as incoming call.
3. Exit - terminates the application or goes back to source phone number input - this is left up to you.

Below is an example of the program workflow, the results shown are different from the ones you will have with a real dataset:

```
Enter your phone number: +123
Phone number not found
Enter your phone number: +79101112233

Select an action:
1 - Call statistics
2 - Call history with other contact
3 - Exit
Your choice: 1
3 outgoing call(s), average time: 4 min 6 sec
3 accepted outgoing call(s), average time: 4 min 6 sec
No accepted incoming calls were found
2 missed call(s)
1 rejected call(s)

Select an action:
1 - Call statistics
2 - Call history with other contact
3 - Exit
Your choice: 2
Enter the target phone number: +123
Phone number not found in the database
Enter the target phone number: +79661121234
Call history with +79661121234:
In: 2020-05-02 15:10:01 - 2 min 10 sec
In: 2020-04-30 09:20:35 - missed
Out: 2020-03-04 10:23:12 - 4 min 20 sec
Call history saved in file +79101112233\_+79661121234.txt

Select an action:
1 - Call statistics
2 - Call history with other contact
3 - Exit
Your choice: 3
<program terminates>
```