PROGRAMMING TECHNIQUES, A.A. 2021/2022

Laboratory 3

Exercise 1 and Exercise 3 are the home assignments that can be optionally submitted for evaluation, to obtain the exam's bonus. Deadline is **8 April 2022**, 11:59 pm, and will be the same for Lab1+Lab2+Lab3. The home assignments of Lab1+Lab2+Lab3 need to be submitted in one shot by the given deadline, following the instructions that are provided in the Portale della Didattica (see the document: Instructions for submission of assigments.pdf)

Objectives

- Solve iterative problems with scalar data (C2 – Problem Solving with Scalar Data, part III: Text processing, Verification)

Technical content

- I/O basics
- Functions
- Conditional and iterative problems
- Elementary manipulation of numbers (int and float) and characters (char)

Exercise 1. **(THIS EXERCISE IS TO BE SUBMITTED FOR THE EXAM BONUS)**
Category: text processing problems

A text file contains only alphabetic characters, numeric characters, punctuation characters (. , ; : ! ? '), spaces and '\n' characters.
Write a C program that generates a second file containing the same text of the input one, modified as follows:

- Replace all the numerical characters with '*'.
- Add a space after each punctuation character (unless it is already followed by a space or a '\n' or another punctuation character).
- Make sure that the first alphabetic character after either a '.', a '!' or a '?' character is uppercase (even if there are spaces or '\n' characters in between). If it is a lowercase character, transform it to uppercase.
- Each line should be of 25 characters maximum, without considering the '\n'. If a line is longer than 25, it needs to be truncated after exactly 25 characters (it does not matter if this splits a word into two).
- To align the text, add spaces at the end of each line that is shorter than 25 (as many they are needed to reach the fixed 25 characters length)
- Each line should end with a printout of the number of characters of the corresponding line in the original file, in the format: " | c:%d \n" (see example below).

The names of the files are constant ("input.txt" and "output.txt") defined with #define.

Example:

Content of input.txt	Content of output.txt
Caratterizzata da un passato turbolento, in epoca medievale Rouen fu devastata piu' volte da incendi ed epidemie e durante la Guerra dei Cent'Anni fu occupata dagli inglesi. nel 1431 nella sua piazza centrale la giovane Giovanna d'Arco (Jeanne d'Arc) fu processata per eresia e arsa sul rogo!durante la seconda guerra mondiale gli Alleati bombardarono ampie zone della citta', soprattutto il quartiere che si estende a sud della cattedrale.	Caratterizzata da un pass| c:25 ato turbolento, in | c:18 epoca medievale Rouen fu | c:25 devastata piu' volte | c:20 da incendi ed epidemie e | c:25 durante la Guerra | c:17 dei Cent' Anni fu occupat| c:25 a dagli inglesi. | c:16 Nel **** nella sua piazza| c:25 centrale la giovane | c:20 Giovanna d' Arco (Jeanne| c:25 d' Arc) fu processata | c:22 per eresia e arsa sul rog| c:25 o! Durante la seconda | c:21 guerra mondiale gli Allea| c:25 ti bombardarono | c:15 ampie zone della citta', | c:25 soprattutto il | c:14 quartiere che si estende | c:25 a sud della cattedrale. | c:23

Exercise 2.
Category: Verification problems with texts - Verification of expressions

A text file expr.txt contains a mathematical expression in each line, consisting of positive integer numbers (unsigned), arithmetic operators (+, -, *, / and %) and opened/closed round brackets.
Write a C program to verify the syntactic correctness of the expressions that are contained in the file, according to the following rules:

1. Brackets need to be balanced: there should be an equal total number of opened and closed brackets, and each closed bracket should correspond to a preceding opened bracket.
2. There cannot be spaces between numbers. On the other hand, spaces are admitted (and to be neglected) if they are placed in any other position within the expression.
3. There should be a correct succession of operands and operators. Each operation should be a sequence consisting of at least two operands, with operators (one of +, -, *, / and %) in between. An operand could be either a number or another expression within round brackets.

In practice, to verify rule 3 you should verify the following:

- After an opened bracket there can only be an operand (in other words, there can only be either a number or another opened bracket). There cannot be an operator.
- After an operand (in other words, after either a closed bracket or a number) there can only be either another closed bracket (if rule 1 is verified), an operator, or the end of the expression.
- After an operator there has to be another operand (that is, either an opened bracket or a number)

Alternatively, to verify rule 3 you can verify the following:

- Before an operand (that is, before either an opened bracket or a number) there can only be an opened bracket, an operator, or the beginning of the expression.
- Before a closed bracket there can only be an operand (that is, either another closed bracket or a number).

- Before an operator there can only be an operand (either a closed bracket or a number).

The program should read the input file and verify the expressions one by one. Whenever the program finds an error in an expression, it should print a message on the screen including the progressive line number that identifies the expression. Then, the rest of the expression should be ignored, and the program should pass to the following one.

Example:

Content of expr.txt	Messages printed on the screen:

`(3 + 2) / 7`
`((3 - 7) 4)`
`(3 - 2) +`
`(8 * 9 1) * 4`
`((4*(2+23)) / (7-3))`
	`Error in expression 2`
	`Error in expression 3`
	`Error in expression 4`

Suggestions: *you may use a counter to keep track of the number of opened brackets, and use two char variables to store the last two characters that were read…*

Exercise 3. **(THIS EXERCISE IS TO BE SUBMITTED FOR THE EXAM BONUS)**
Category: verification of numerical sequences

A text file (named numbers.txt, defined with #define) reports a sequence of integer numbers, with either spaces or '\n' as separators. Write a C program to verify that each i-th value x_i (with $i \geq 2$) can obtained by the previous two values (x_{i-2} and x_{i-1}) by applying a single arithmetic operator. In other words, you should verify that each x_i is either the sum ($x_{i-2} + x_{i-1}$), the difference ($x_{i-2} - x_{i-1}$), the product ($x_{i-2} * x_{i-1}$) or the division (x_{i-2} / x_{i-1}) of the previous two (NB careful to avoid divisions by 0!).
If a value does not verify the acceptance criteria, the program needs to discard it and pass to the next one. The program should also identify the maximum and minimum values of the sequence, excluding the non-valid data from the computation.

At the end of the verification, the program should print on the screen:
1. the final result of the verification. In case the sequence contained only valid values, the program should print a corresponding message. Otherwise, it should print how many values have been discarded.
2. the maximum and minimum values of the sequence, excluding the discarded values from the computation.

Example

Content of numbers.txt	Messages on the screen	Comment

`12`
`3`
`4`
`7 -3`
`0`
`4`
`1`
`3`
`3 9`
`11`
	`# of discarded values: 2`
	`Maximum value: 12`
	`Minimum value: -3`
		The program should discard the following values:
		• 0 (it cannot be obtained by any operation on 7 and -3)
		• 11 (it cannot be obtained by any operation on 3 and 9)