PROGRAMMING TECHNIQUES, A.A. 2023/2024
Laboratory 2

Objectives

- Solve **iterative** problems on **scalar** data: numeric problems and encoding problems (C2 – Problem Solving with Scalar Data, parts I and II)

Technical content

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

Exercise 1.

Skills: Formatted IO, manipulation of numbers
Category: numerical problems, problems of numerical encoding

Write a C program that, after acquiring 2 positive **integer** numbers **A** and **B** from keyboard, computes the greatest common factor (GCF) of A and B by applying the Euler's method.

Euler method (remainders method): the method repeatedly applies divisions of the greater number by the smaller one, replacing at each step the greater value with the smaller and the smaller with the remainder of the division, respectively. The iterations end when the remainder is 0.

Example: **A** = 34, **B** = 18

iteration 1: 34 **%** 18 = 16
iteration 2: 18 **%** 16 = 2
iteration 3: 16 **%** 2 = 0 ← **stop!**

Result: GCF = 2

Exercise 2.

Skills: Formatted IO, manipulation of numbers
Category: numerical problems

Write a C program that prints on the screen the first **N** numbers of Fibonacci series (**N** should be acquired from keyboard).

Suggestion: Fibonacci series is 0 1 1 2 3 5 8 … It can be developed by applying the following:

$$\mathbf{X}_i = \mathbf{X}_{i-1} + \mathbf{X}_{i-2}, \text{ where } \mathbf{X}_0 = 0 \text{ and } \mathbf{X}_1 = 1.$$

In-depth

Modify the series as follows: $\mathbf{X}_i = \mathbf{X}_{i-1} * \mathbf{X}_{i-2}$, with $\mathbf{X}_0 = 1$ and $\mathbf{X}_1 = 2$

Try to determine experimentally (i.e., by analysing the results that you obtain with increasingly larger number of elements of the series) how many elements can be represented without error if you use **int** variables or if you use **unsigned int** variables, respectively.

Exercise 3.

Skills: IO on files, manipulation of characters
Category: encoding problems with texts/characters

A text file (for example: `source.txt`) contains a text with an unknown number of lines. Assume that the file DOES NOT contain the character '**!**'.

The aim of the program is to compress the file, by replacing any sequence of repeated characters (with a minimum of 2 and a maximum of 9 repetitions) with the following triplet of characters:

<**repeated character**>**!**<**number of repetitions**>.

NB: the number of repetitions should not consider the first occurrence of the character: for example, **AA** contains **1** repetition, **BBB** contains **2** repetitions, etc.
In case a character is repeated more than **9** times, the sequence should be broken into multiple sub-sequences. For example:
- "**AAAAAAAAAAAAAA**" should be encoded as "**A!9A!3**"
- "the number **100000000** is large" should be encoded as "the number **10!7** is large"
- "there are 15 repeated = : =============== and 4 dots…." should be encoded as "there are 15 repeated = : **=!9=!4** and 4 dots**.!3**"

The compressed text should be stored in a second file (for example: `compressed.txt`).

Example:
If input file `source.txt` is:
```
Partenza      Destinazione    Costo
  Parigi          New York     1000
     Roma            Londra      700
   Sidney        Los Angeles     2222
```

The file `compressed.txt` should be:
```
Partenza !5Destinazione !3Costo
  Parigi !9New York !410!2
 !4Roma !9  Londra !5700
 !2Sidney !6Los Angeles !42!3
```

Write **two different functions**, respectively able to perform the ***compression*** (from original to compressed encoding) and the ***decompression*** (reverse operation: from compressed to original encoding). The prototypes of the two functions should be

```
int compressing(FILE *fin, FILE *fout);
int decompressing(FILE *fin, FILE *fout);
```

In case of error, the functions should return 0. In case of successful operation, they should return the number of characters that were printed on the output file.

The **main program** should:
- allow the user to decide (with a corresponding input from keyboard) whether to perform a **compression** or a **decompression** operation
- open the input and output files

- based on the decision of the user, call either the compressing or the decompressing function on the opened files.

Suggestion: in your experiments, use a third filename (for example: source2.txt) for the output of the decompression, so that you won't overwrite the original source.txt

Exercise 4.

Skills: IO on files, manipulation of characters.
Category: encoding problems with texts/characters

A file contains a text with an unknown number of characters. The aim of the program is to create a second file where the characters of the original file are re-encoded as follows.

Numeric characters:

- Numeric characters ('**0**'..'**9**') should be re-encoded into the numeric character that is **k** positions later in the **ASCII** table, with **k** starting from **0** and incremented by **1** each time a new numeric character is re-encoded. (**NB**: it is a **10 MODULE**: the count starts again from **0** after **9**).
  For example, if the file starts as follows: "Number 248 is even":
  - '**2**' (**k** starts from **0**) should be re-encoded as '**2**'+**0** = '**2**' (**k** becomes 1)
  - '**4**' should become '**4**'+**1** = '**5**' (**k** becomes **2**)
  - '**8**' should become '**8**'+**2** = '**0**' (after '**9**' we start again from '**0**').

Alphabetic characters:

- If the alphabetic character is preceded by a non-alphabetic character, it should remain unchanged
- If it is preceded by another alphabetic character (**c0**), the code should be incremented by **h** positions within the set of alphabetic characters (with **h** = **c0**-'**A**' if **c0** is uppercase, **h**=**c0**-'**a**' if **c0** is lowercase). The h increment should be a **26 MODULE** (that is, after reaching the code of '**z**' or '**Z**' (respectively in case of lowercase or uppercase characters) we start again from '**a**' or '**A**'.

The encoding result should be stored in a second file (the names of the input and output files are acquired from keyboard).

Example
If input file is:

```
Apelle figlio di Apollo
fece una palla di pelle di pollo
tutti i pesci vennero a galla
per vedere la palla di pelle di pollo
fatta da Apelle figlio di Apollo.
```

The output file `encoded.txt` should be:

```
Aptept fntema dl Apdozn
fjlp uhh ppall dl ptept dl pdozn
tngzh i ptlnv vzmzdui a ggrcc
ptk vzcgxb ll ppall dl ptept dl pdozn
ffyrr dd Aptept fntema dl Apdozn.
```

Write **two different functions**, respectively able to perform the ***encoding*** (from original source file to encoded) and the ***decoding*** (reverse operation: from encoding to original). The prototypes of the two functions should be

```
int encoding(FILE *fin, FILE *fout);
int decoding(FILE *fin, FILE *fout);
```

In case of error, the functions should return 0. In case of successful operation, they should return the number of characters that were printed on the output file.

The **main program** should:

- allow the user to decide (with a corresponding input from keyboard) whether to perform an encoding or a decoding operation
- open the input and output files
- based on the decision of the user, call either the encoding or the decoding function on the opened files.