PROGRAMMING TECHNIQUES, A.A. 2023/2024
Laboratory 8

Objectives
- Understand the use of pointers and their representation

Tecnical contents
- Basics of Input Output
- Use of functions
- Conditional and iterative constructs
- Elementary manipulations of **vectors** and **matrices** (of int and float)
- Pointer data type
- Passing parameters "by pointer"

Exercise 1.
Skills: reading / writing files, manipulation of matrices, pointers and passing parameters by reference (i.e., "by pointer")
Category: verification and selection problems (Problem solving with arrays - complex problems) – Pointers data type (Lecture C5)

Detection of black regions (with parameters passed by reference)

Re-implement exercise 1 of Laboratory 7 as follows.

- Supposing that you have declared in the main a matrix of integers `M` of dimension `NR x NC`, with `NR` and `NC` equal to 50, the acquisition of the content of the matrix from the text file should be handled by a function `readMatrix`, that should be able to return the actual number of rows and columns of the acquired matrix to the caller "by reference" (or better, "by pointer"). This function may have the following prototype:

  ```
  void readMatrix(int mat[][NC], int maxR, int* nrp, int *ncp);
  ```

  and may be called by the main with the following instruction:

  ```
  readMatrix(M,NR,&nr,&nc);
  ```

  As a result of the function call, the integer variables `nr` and `nc` should contain the actual number of rows and columns of the acquired matrix, `M`.

- The detection of regions should be handled by a function `detectRegion` that, given a box of the matrix identified by row and column indexes `r` and `c`, should determine whether this box represents the upper left corner of a rectangular region. The function should return a Boolean integer as the return value: respectively, 1 if the region is found, 0 otherwise. The dimensions (width and height) of the rectangle should also be returned to the caller "by reference", as it was done for `nr` and `nc` in the previous point. The function should be called as follows:

```
if (detectRegion(M,nr,nc,r,c,&w,&h)) {
   … … …
   // print a message corresponding to the rectangle
   // having upper-left corner (r,c), width w and height
}
```

Exercise 2.
Skills: pointers, representation of the information), numerical representations (from Computer Sciences course)
Category: pointers data type (lecture C5)

Pointers and data representation

Realize a program that allows to visualize the internal (binary) encoding of a real number, realized in C with either a float or a double data type.

In C, float and double types (see the definition in https://en.wikipedia.org/wiki/C_data_types) implement the IEEE-754 specifications for real data types in single and double precision, respectively. You should remember the IEEE Standard for Floating-Point Arithmetic (IEEE 754) from the Computer Sciences course. You can find all the details here:
https://en.wikipedia.org/wiki/IEEE_754
https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/

The C program should:

- Declare two variables for real values (`af` and `ad`, respectively of float and double type)
- Verify whether the computer uses the little Endian or the big Endian encoding technique, and based on this assign a 1/0 value to an integer variable `bigEndian` (respectively, 1 if the encoding technique is big Endian, 0 otherwise). See the HELP section for some suggestions on how to do this.
- Visualize (using the operator `sizeof)` the dimension (in byte as well as in bit) of `af` and `ad`
- Acquire a real value from keyboard (with decimal point and, eventually, an exponent in base 10) and assign it to the two variables `af` and `ad`
- Calling a function `printEncoding`, print the internal binary encoding of the number in the two variables `af` and `ad`

The function `printEncoding` should have the following prototype:

```
  void printEncoding (void *p, int size, int bigEndian);
```

and should be called twice (respectively, on the float and on the double variable), as follows:

```
printEncoding((void *)&af,sizeof(af),bigEndian);
printEncoding((void *)&ad,sizeof(ad),bigEndian);
```

where the three parameters passed to the function are: **1)** the pointer to the variable (converted to a "generic" pointer type `void*` with a cast), **2)** the size of the variable, and **3)** the `bigEndian` flag, that provides information about the type of endianness of the encoding.

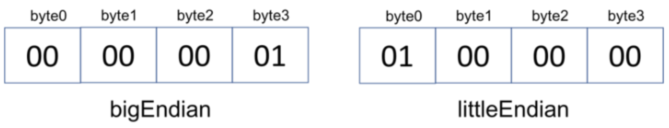
By applying the pointers arithmetic, the function should print on the screen the internal binary encoding of the variable (respectively: the bit of the sign, the bits of the exponent and the bits of the mantissa).

See the following page in case you need some help/suggestions.

-- HELP --

How to check the type of endianness of the system??

You may declare an integer variable `test`, initialized to a value of your choice (for example: the value 1, that in base 16 is `0x00000001`). Assuming that int variables are stored in 4 bytes (you can check with `sizeof)`, big Endian and little Endian encoding are respectively as follows:



where byte0 is the cell of 1 byte with the smallest address. Hence, one just needs to check what is stored in byte0: the encoding is big Endian if the content of byte0 is 0, little Endian if it is 1.

Now…how to access the value of the individual bytes of an integer variable? You need to understand and use pointers arithmetic!
Remember that `test` is of **`int`** type, that is a 4-byte datum. As a consequence, `&test` is a pointer to **`int`**, which can only be used to access the content of 4-byte data. To handle cells of 1 byte, you will need to use a pointer to a datum of 1 byte!
As you know, **`char`** data type has 1 byte in size. Then, you can declare a pointer to char, `pchar`, and you can initialize `pchar` to the same address pointed by `&test`, as follows:

```
char *pchar = (char *)&test;
```

(note that the assignment requires that the two pointers are of the same type: hence, `&test` is converted to `char *` type with a cast operator).
By doing so, you will obtain that `pchar` points to the first byte of the integer datum `test` (byte0, in the figure), `pchar+1` points to byte1, `pchar+2` to byte2, etc. Hence, the content of byte0 is `*pchar`, the content of byte1 is `*(pchar+1)`, etc.

Some general suggestions to solve the exercise

Note that you are NOT asked to represent the exponent and the mantissa as numbers, but only to display the bits. Although you can try different solutions, we suggest the following strategy:

- since it is not possible to create a vector of "bits" in C, you may read and represent the real number as a vector of `unsigned char`, where an `unsigned char` in C is a non-negative 8 bits value, in the 0 to 255 range. For example, a 32-bit `float` corresponds to a 4 bytes representation. Hence, it will correspond to a vector of 4 `unsigned char`.
  Each element of the vector must then be decoded by means of a binary conversion algorithm (that is, an algorithm that recognizes the bits of an unsigned number).
  In the practice, the pointer p (of type void *) of the function `printEncoding` can be internally assigned to a pointer to `unsigned char`…

- the bits must be printed starting from the most significant one (MSB), separating the sign, exponent and mantissa bits. Depending on the `bigEndian` parameter, you know the direction in which the bytes of the number should be evaluated. The `size` parameter is used to know where the exponent bits end and the mantissa bits begin. Scanning the bytes can be carried out by properly applying pointer arithmetic, such as to go through all the bytes from the most significant to the least significant (or vice versa).