Programming for Data Science (Full exam 15/07/2024)

Upload the solutions to the programming exercises to the following link:
https://evo.di.unipi.it/student/courses/16/exams/E07y8J3

Exercise 1. (Math, on paper)

A. Determine how many bit-strings of length 8 either start with 1 or start with 01. Determine how many bit-strings of length 6 contain exactly 3 0s.
B. For which values of k and m, respectively, the following matrices are singular

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & k \end{pmatrix} B = \begin{pmatrix} 1 & 2 & 3 \\ 0 & m & 4 \\ 3 & 1 & m \end{pmatrix}$$

C. Consider (P({a,b,c}),⊆), where P({a,b,c}) represents the power set of the set {a,b,c}, and ⊆ denotes set inclusion.
C1. Represent the partial order with a Hasse diagram.
C2. Is (P({a,b,c}),⊆) a lattice?
C3 . Define (a graph is fine) a total ordering (P({a,b,c}),≼) compatible with the partial ordering (P({a,b,c}),⊆)

Exercise 2. (Python) In mathematics, Pascal's triangle is a triangular array of the binomial coefficients. The rows of Pascal's triangle are conventionally enumerated, starting with row n = 0 at the top (the 0th row). The entries in each row are numbered from the left, beginning with k = 0, and are usually staggered relative to the numbers in the adjacent rows. The triangle may be constructed in the following manner: in row 0 (the topmost row), there is a unique nonzero entry 1. Each entry of each subsequent row is constructed by adding the number above and to the left with the number above and to the right, treating blank entries as 0.

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5   10  10  5   1
    1   6   15  20  15  6   1
  1   7   21  35  35  21  7   1
```

Define a Python function PascalT(n) that takes as input an integer n and visualises the first n rows of Pascal's triangle. Test your function by asking the user for an integer and calling the just-defined function on this integer.

Exercise 3. (C) Write a C function called removeNegEven() that **takes in input** an array of integers vect, its size dim, and a pointer to int called new_dim. The function modifies the array vect as follows: for each occurrence a of an even number in vect, (i) it replaces a with its absolute value; and (ii) it adds at the end of the array an element containing the absolute value of a. For instance, assuming that a is at position i of vect, we will have that vect[i] will be a positive (even) number, and a new element will be added at the end

of vect, containing the absolute value of vect[i]. The insertion of the elements at the end of the array must follow the order of the corresponding even numbers in vect.
The function **returns** the new array after the resize, and writes the size of the resized array at the address pointed by new_dim.

Example: [4 -2 2 -3] → [4 2 2 -3 4 2 2]

Test the function in the main(), as follows:

1. Ask the user to input an integer n, which will be the size of an array;
2. **Dynamically** create an array arr of size n, and populate it with random integers between -150 and 150;
3. Print the content of the array arr as follows: the elements are all printed on the same line, separated by a single space (add a newline at the end);
4. Call the function removeNegEven() on the array arr;
5. Print again the content of the array arr (that at this time should have been modified by removeNegEven()).