Programming for Data Science (Full exam 03/06/2024)

Upload the solutions to the programming exercises to the following link:
https://evo.di.unipi.it/student/courses/16/exams/o0JGR4V

Exercise 1. (Math, on paper)

A. Binary numbers:
   a. Add the binary numbers 101101_2 and 11011_2 (express your answers in binary form)
   b. Convert the following decimal number to its binary equivalent: 45

B. Let's consider a propositional language where: p means "Paola is happy", q means "Paola paints a picture". Formalize the following sentences:
   a. "if Paola is happy, then she paints a picture"
   b. "Paola is happy only if she paints a picture"

C. Let (P, ≤) be the partially ordered set (poset) defined by:
   P = {2, 4, 5, 6, 7, 8, 10, 24}
   a ≤ b if and only if a divides b, i.e., b is a multiple of a
   a. Draw the Hasse diagram of (P, ≤).
   b. Find all the upper bounds and lower bounds of {2, 4, 6}.

Exercise 2. (Python) Create a Python class `CustomSet` that mimics the behavior of a set using only the list data structure. Implement the following methods:

- **Add Element**: Implement a method `` `add_element` `` to add an element to the set. Ensure that duplicate elements are not added.
- **Remove Element**: Implement a method `` `remove_element` `` to remove an element from the set. Handle the case where the element is not present in the set.
- **Contains**: Implement a method `` `contains` `` to check if an element is in the set. Return True if the element is present, otherwise return False.
- **Intersection**: Implement a method `` `intersect` `` that takes another CustomSet as an argument and returns a new CustomSet containing only elements in both sets.
- **Union**: Implement a method `` `union` `` that takes another CustomSet as an argument and returns a new CustomSet containing all elements from both sets without duplicates.

Instructions:

- Define a class CustomSet with an attribute `` `elements` `` that is a list.
- Implement the methods add_element, remove_element, contains, intersect, and union.
- Ensure proper validation and error handling in each method.
- Write a helper method for pretty printing the content of a CustomSet object (with the print statement)
- Demonstrate the usage of the CustomSet class with example operations.

Exercise 3. (C) Write a C program that implements the same `CustomSet` data structure as highlighted in Exercise 2, by exploiting a struct named `` `CustomSet` `` memorizing the elements in a dynamic array that doubles in size when the array is full and halves in size when its occupancy ratio falls below 20%. The array doubling and halving ensure efficient memory usage as the CustomSet grows and shrinks. The array capacity starts from 4, and never falls below that size. Implement the several methods described in Exercise 2 for working with the `CustomSet`. Within the main, create a `CustomSet` and test the implementation of each method.