Programming for Data Science (04/09/2023)

Name files as "<your matricola>_<firstname>_<lastname>_ex2.py" for Exercise 2, and "<your matricola>_<firstname>_<lastname>_ex3.c" for the third exercise.

Upload the TWO files in a folder
(named with your student number and your last name) at the following URL: *Upload here*
(access GDrive using your university credentials)

Exercise 1. (Math, on paper)

Let A be a 3x3 upper triangular matrix, i.e. all the entries below the main diagonal are zero:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{bmatrix}$$

1) Write a logical formula of the kind $\forall i, j.$ ….. telling that a square matrix is upper triangular.
2) The sum of two upper triangular matrices is always: (provide an explanation)
    a) Upper triangular
    b) Lower triangular
    c) Diagonal
    d) Symmetric
    e) None of them
3) The product of two upper triangular matrices is always: (provide an explanation)
    a) Upper triangular
    b) Lower triangular
    c) Diagonal
    d) Symmetric
    e) None of them
4) What is the determinant of A? Show how you computed
5) Which are the eigenvalues of A? Show how you computed

Exercise 2. (Python) Define the following Python functions

1) `is_upper_triangular(matrix)` that takes a square matrix as input and returns `True` if the matrix is upper triangular, and `False` otherwise
2) `det_if_triangular(matrix)` that takes a square matrix and, if it is upper triangular returns its determinant
3) `tra_if_triangular(matrix)` that takes a square matrix and, if it is upper triangular returns its transpose (without using the Python primitives for transpose)
4) Write a Python function `solve_upper_triangular_system(matrix, vector)` that takes an upper triangular matrix and a vector as input, and solves the system of linear equations using back-substitution.
   Hint: Begin by solving for the last variable in the system (by multiplying the last row of the matrix and the vector). Substitute the value of the solved variable back into the second-to-last equation. Continue this process …

Exercise 3. (C) Write a C program that implements a linked list using dynamic memory allocation. In particular:

1) Create a struct `Node` to represent a single node in the linked list. Each node should have an integer data field and a pointer to the next node.
2) Create a struct `LinkedList` to represent the linked list itself. The struct should have a pointer to the head node (the first node in the list).

Then implement the following operations for the linked list:

1) `insertFront(LinkedList* list, int data)`: Insert a new node with the given data at the front of the list.
2) `insertEnd(LinkedList* list, int data)`: Insert a new node with the given data at the end of the list.
3) `delete(LinkedList* list, int data)`: Delete the first occurrence of a node with the given data from the list.
4) `display(LinkedList* list)`: Display the elements of the linked list from the head to the end.

Finally, write a main function to test the linked list operations. Create a linked list, insert some elements, delete elements, and display the list after each operation to verify the correctness of your implementation.