HOMEWORK 2

Write a **C program** that may **solve linear equations systems** according to the instructions below. The **code template** that also includes commentary (which may be removed while implementing) for the program is provided in **solver.c** file and the program may be implemented over the provided template.

- A **Matrix** structure is provided in the code template. Matrix structure consists of three members: a double pointer to **float **M** for a 2-dimensional array (that will be referred to as the **matrix** for the rest of this document) and two **integer** variables **n** and **m** for the numbers of rows and columns of the matrix respectively. Matrix structure should be used for matrices throughout the program.

The function prototypes given below need to be implemented.

- **Matrix construct(Matrix x, int n, int m)**; function should return the Matrix x argument back after allocating a memory for n×m sized matrix using its pointer member via **malloc()** or **calloc()** functions and assigning n and m arguments to its integer members respectively.
- **Matrix destruct(Matrix x)**; function should return the Matrix x argument back after properly deallocating the memory allocated for its pointer member via **free()** function.
- **Matrix configure(Matrix x)**; function should return the Matrix x argument back after reading rows and columns numbers from the user input into its integer members, allocating memory for its pointer member via **construct()** as per rows and columns numbers, and reading the matrix element values based on a matrix format. While reading user inputs, the function should first print the instruction that "**Enter the matrix size as numbers of rows and columns:** " and read two inputs in integer type into the corresponding integer members of Matrix x argument at once. Then, it should print the instruction that "**Enter the matrix elements row by row:** " and read each matrix element input in float type one-by-one into the pointer member of Matrix x argument using a 2-dimensional matrix indexing.
- **void print(Matrix x)**; function should print value of each element in the matrix held by the pointer member of the Matrix x argument in a table format, row-by-row and separating the columns by tab character.
- **float determinant(Matrix x)**; function should return the calculated determinant value for the Matrix x argument. The matrix of Matrix x argument should be checked if it has a square matrix or not. If it is a non-square matrix (if row and column numbers unequal), then an error message that "**The matrix is non-square and has no determinant!**" should be printed (In this case, the determinant value will stay uninitialized at the end of function.). If it is a square matrix but its size is smaller than 1 (i.e. 0×0), then an error message that "**The matrix does not exist!**" should be printed and the function should return immediately (In this case, the determinant value is left uninitialized.). If it is a 1×1 square matrix, then the determinant value should be equal to the value of the only element in the matrix. If it is a 2×2 square matrix, the determinant value may be obtained by subtracting the product of secondary diagonal elements from the product of primary diagonal elements. If it is a square matrix and its size is larger, then the determinant value should be calculated by defining and constructing **a Matrix for minors** via **construct()** function and using **recursion** as per (1). See https://mathworld.wolfram.com/Determinant.html and https://mathworld.wolfram.com/DeterminantExpansionbyMinors.html.

$$|\mathbf{x}_{n\times n}| = \sum_{j=0}^{n} (-1)^{i+j} \times \mathbf{x}_{ij} \times \left|\mathbf{minor}_{\mathbf{x}_{ij}}\right| \quad \text{where } i = 0 \text{ (iterate only for the first row)} \qquad (1)$$

- **Matrix cofactor(Matrix x)**; function should return a Matrix that its matrix holds the cofactors of the Matrix x argument. **The Matrix defined for cofactors** should be constructed with the same size as Matrix x argument via **construct()**. The matrix of Matrix x argument should be checked if it has a square matrix or not. If it is a non-square matrix (if row and column numbers unequal), then an error message that "**The matrix is non-square and has no cofactor!**" should be printed (In this case, the cofactor matrix elements will stay uninitialized at the end of function.). If it is a square matrix but its size is smaller than 1 (i.e. 0×0), then an error message that "**The matrix does not exist!**" should be printed and the function should return immediately (In this case, the cofactor matrix elements are left uninitialized.). If it is a 1×1 square matrix, then the only element in the cofactor matrix should be equal to 1. If it is a square matrix and its size is 2 or larger, then the cofactor matrix elements may be considered as the signed minors. Therefore, they should be calculated by defining and constructing **a Matrix for minors** via **construct()** and using **determinant()** function as per (2).

$$\mathbf{cofactor}_{ij} = (-1)^{i+j} \times \left|\mathbf{minor}_{\mathbf{x}_{ij}}\right| \quad \forall i, j \in \mathbb{N} \text{ and } i < n, j < m \tag{2}$$

- **Matrix transpose(Matrix x)**; function should return a Matrix that its matrix is the transpose of the matrix of the Matrix x argument. **The Matrix defined for transpose** should be constructed with a size of the swapped numbers of rows and columns of the matrix of the Matrix x via **construct()** function. Row elements of the matrix of the Matrix x argument should be assigned to column elements of the transpose matrix.

- **Matrix inverse(Matrix x)**; function should return a Matrix that holds the inverse of the matrix of Matrix x argument. **The Matrix defined for inverse** should be constructed as a square matrix with a size of the number of rows of the matrix of Matrix x argument via **construct()** function. The determinant value for the matrix of Matrix x argument should be calculated via **determinant()** function. If the determinant is 0, then an error message that "**The matrix is non-square or singular and has no inverse!**" should be printed (In this case, the inverse matrix elements will stay uninitialized at the end of function.). the determinant value is nonzero; this means that the matrix of Matix x is square and has a nonzero size. Therefore, the inverse matrix should be calculated by defining and constructing **a Matrix for adjoints**, which is the transpose of the cofactor matrix via **transpose()** function and the inverse matrix elements should be calculated as per (3). After the calculation finished, the adjoint matrix may be deallocated via **destruct()** function.

$$\mathbf{x}_{ij}^{-1} = \frac{\mathbf{adjoint}_{ij}}{|\mathbf{x}_{n\times n}|} \quad \text{where } \mathbf{adjoint}_{n\times n} = \mathbf{cofactor}_{n\times n}^{T} \text{ and } |\mathbf{x}_{n\times n}| \neq 0 \quad \forall i, j \in \mathbb{N} \text{ and } i, j < n \tag{3}$$

- **Matrix multiply(Matrix x, Matrix y)**; function should return a Matrix that its matrix holds the multiplication of the matrix of Matrix x argument with the matrix of Matrix y argument. **The Matrix defined for matrix multiplication** should be constructed with a size of the number of rows of the first argument matrix × the number of columns of the second argument matrix via **construct()** function. If the number of columns of the first argument matrix equals the number of rows of the second argument matrix, then each element of the multiplication matrix should be equal to the sum of products of the corresponding row elements of the first argument matrix and the corresponding column elements of the second argument matrix. Otherwise, an error message that "**Sizes of matrices do not match for the matrix multiplication!**" should be printed (In this case, the multiplication matrix elements will stay uninitialized at the end of function.).

- **Matrix solve(Matrix A, Matrix b)**; function should return the solution in the Matrix type for a linear equations system specified by $\mathbf{Ax} = \mathbf{b}$ as directly the result of $\mathbf{A}^{-1}\mathbf{b}$ operation via **inverse()** and **multiply()** functions.

The program should be able to solve a well-defined linear equations system $\mathbf{Ax} = \mathbf{b}$ where the coefficient matrix $\mathbf{A}_{n\times n}$ and the constant (right-hand side) vector $\mathbf{b}_{n\times 1}$ are known system parameters, and the variable vector $\mathbf{x}_{n\times 1}$ is the system solution that needs to be solved.

- **The main() function**, as written in the code template **solver.c** already, should start with the definition and the configuration of Matrix A and Matrix b. Matrix A and Matrix b should be configured with proper and compatible sizes via **configure()** function. Then, Matrix x should be defined and initialized by the system solution that is obtained via **solve()** function. The function and the program should end with printing the obtained result.

Note:

All functions are given in a flow that allows one to utilize each implemented function in the next one. Even the **print()** function that is not called in the final code may help to test and investigate the working behaviors of all other functions in the intermediate implementation steps. Thus, it is strongly recommended to implement these functions in the given order.

IMPORTANT NOTES

1. The **due date** of this **Homework 2** is **24 DECEMBER 2023** Sunday at **23.55 sharp**.
2. This is a **group assignment**, and each group may consist of **2 members** at most.
3. Each group should submit a **single *.c file** (preferably implemented over the provided code template **solver.c**). Other file types will neither be accepted nor graded.
4. Only **one group member** should **submit** the requested file on behalf of the group. Therefore, both group members' **full names** and **student numbers** must be specified on the top lines of the submitted *.c file as comments.
5. Submitted *.c file should be **clean and easy to read**. Explanatory **comments** should be provided anywhere it is required. All variables and functions (if needed) must comply with **the naming conventions** logically and formally. See https://en.wikipedia.org/wiki/Naming_convention_(programming).
6. Submitted *.c file needs to be **compiled** without any problem, otherwise the **grades** for those files will be cut.
7. Each functionality will be tested and graded.
8. Late submissions will reduce **20 points** per each day late.
9. **Cheating is strictly forbidden**! Once cheating is detected, all involved submissions will suffer the consequences (i.e., negated grades).