Institute for Advancing Intelligence, TCG CREST
(TCG Centres for Research and Education in Science and Technology)

Introduction to Programming and Data Structures
Ph.D. Coursework: First year, First Semester (Session: 2024-25)
Assignment #09

Full Marks: 200 — Instructor: Dr. Laltu Sardar
Clarification Deadline: **2024-Dec-05** — Submission Deadline: **2024-Dec-08**

Instructions:

1. Keep all functions in "`matrix.h`" file, and create a separate test file named "`matrix_test.c`".
2. The program should be as fault-tolerant as possible, handling potential input errors gracefully.
3. The test file should include a menu, and input matrices must be provided via files only.
4. You must use previously defined matrix-related functions, e.g., matrix scanning, allocation, etc.

Problem #0901: Advanced Matrix Algorithms

Implement the following functions in C.

1. `float matrixDeterminant(Matrix A, int n);`: This function calculates the determinant of a square matrix A of size $n \times n$. It uses a cofactor expansion approach for small matrices (like 3x3), which provides a recursive and straightforward method to compute the determinant. The function returns the determinant as a `float` value. If the matrix is singular (determinant is zero), the matrix has no inverse. [50]
2. `Matrix matrixInverse(Matrix A);`: This function calculates the inverse of a square matrix A if it is non-singular. The function returns a new `Matrix` struct that contains the elements of the inverse matrix. It first calculates the determinant to check for singularity and then finds the adjugate of A, dividing each element by the determinant to yield the inverse matrix. [70]
3. `void dominantEigen(Matrix A, float *eigenValue, EigenVectors *evs);`: This function computes the dominant eigenvalue and its corresponding eigenvector of the matrix A using the power method. It takes the matrix A and a pointer to store the `eigenValue` as a `float`. The `EigenVectors *evs` parameter is a pointer to an `EigenVectors` struct, which will store the dominant eigenvector in the `evc` array. The function iteratively multiplies an initial vector by the matrix and normalizes it until convergence to the dominant eigenvector direction. [80]

Data Structures

- `typedef struct matrix_t{int row, col; float ** arr;}Matrix;`
  This struct defines a matrix data type. It includes:
  - `int row`: Number of rows in the matrix.
  - `int col`: Number of columns in the matrix.
  - `float **arr`: Pointer to a 2D array of floats that stores the matrix elements.
- `typedef struct eigenvectors_t{int col; float *evc;}EigenVectors;`
  This struct defines a vector data type to store eigenvectors. It includes:
  - `int col`: Size of the eigenvector (typically equal to the number of columns in matrix A).
  - `float *evc`: Pointer to an array of floats that stores the components of the eigenvector.