MATRIX CALC v1.0
Ibrar Yunus
BSCS-2B
REG#1190

Software Documentation

This software can be cloned from the github repo as:

$ git clone https://github.com/IbrarYunus/Advanced_Programming.git

Or via web :
https://github.com/IbrarYunus/Advanced_Programming/tree/master/IbrarYunus_Lab1%5Bv1%5D

The project file consists of two parts:
The implementation (in MatrixCalc.c) contains the functions to perform matrix calculations and other utilities. These are:

1. **filer()**
   - tries to open 'file.txt' for matrices, it reads the file line by line (each line has a matrix) and saves each to the struct array MATRIX[].
2. **Operation_Choose(struct matrices mat1, struct matrices mat2, char choice)**
   – takes two parameters for matrices and one other for the operation of choice (these can be +, - , *) . This function also compares rows and columns of the matrices to see if the required operation can be performed or not and then may call the required function.
3. **Subtraction(int **mat1, int **mat2, int rows, int cols)**
   - Called by Operation_Choose(). This takes two matrices and the rows and column of the first matrix as parameters. It performs element-size subtraction and stores the result in global variable subtractionAnswer.
4. **Addition(int **mat1, int **mat2, int rows, int cols)**
   - Called by Operation_Choose(). This takes two matrices and the rows and column of the first matrix as parameters. It performs element-size addition and stores the result in global variable addiitonAnswer.
5. **Multiplication(int** mat1, int** mat2, int m, int q , int p)**
   - Called by Operation_Choose. This takes two matrices and the rows of first matrix and column of the second matrix as parameters. Performs matrix multiplication and stores the result in global variable multiplyAnswer.
6. **PrintMat(int **mat, int rows, int cols)**
   - Takes a matrix, and its number of rows and columns as parameters. Prints the matrix to stdout in a human readable format.
7. **CreateNewMatrix(int rows, int cols)**
   - Utility to dynamically reserve memory for a matrix (i.e. rows and columns read at runtime from the file)

Static Part
The static portion consists of:
-Maximum matrices read from the file can be 300 (as limited by “struct matrices MATRIX[300] “)
-A line in the text file can have maximum size of 800 bytes/characters (as limited by char str[800])
-An addition, subtraction or multiplication operation temporarily stores answer in these variables:

struct matrices multiplyAnswer;
struct matrices addiitonAnswer;
struct matrices subtractionAnswer;

-hence complex operations have to be broken into parts.

Unit Testing Utility
The unit tests are designed to be automatic and uses assertions to compare verified answers to the results of matrix operations from functions of MatrixCalc.
Hence, if the first 5 rows from the file are modified/deleted the assertions will fail (Though the result from the matrix operation may still be correct).

Complex matrix computations are broken into steps (as Operation Choose takes only two parameters). This is by design, as C language does not have concepts as 'operator overloading' and 'classes'.

Sample File Structure
The following line read from the file,

ROWS:2 COLS:2 1 2 3 4 ;

Will create a matrix as:

1 2
3 4

The ';' character signifies the end of a line.

Important NOTE:

If the file contains five matrices, i.e.:

ROWS:2 COLS:2 1 2 3 4 ;
ROWS:3 COLS:3 1 2 3 4 5 6 7 8 9 ;
ROWS:2 COLS:2 5 7 5 4 ;
ROWS:2 COLS:2 2 2 3 2 ;
ROWS:2 COLS:3 1 1 1 1 1 1 ;

This will cause memory for 5 elements of “struct matrices MATRIX[300]” to be reserved in memory and populated by the values from the file.
Hence you can perform operations with MATRIX[0] to MATRIX[4] as the rest of the elements of “struct matrices MATRIX[300]” will not be referring to any memory region.