ACS130 Introduction to Systems Engineering and Software
C Programming Assignment ACS130-004

Tara Baldacchino (t.baldacchino@sheffield.ac.uk, Room D13, AJB)

Assignment weighting: 8% of module mark

Assignment released: Friday 20 March 2020 (Semester 2, week 6)

Assignment due: 11.59pm Monday 13 April 2020.

Submission: You must submit your .c file to the Blackboard (MOLE) assignment dropbox entitled. "ACS130-005" in the Assessment section. Do not copy your .c file into any other format. Do not submit executable files to the Blackboard (MOLE) assignment dropbox. **Multiple submissions to the Blackboard (MOLE) assignment dropbox will be allowed. The version you submit last is the version that will be marked.**

How the marking and feedback will be done
Given the current situation, the module leader will mark your code. You must make sure that your code runs before submitting. Any code that won't run properly, will be marked as is which could mean you could get a 0. **It is your responsibility to submit code that compiles and runs.**

You will receive written feedback and a mark for your code by the end of week 10. Provisional marks may change as a result of unfair means and/or late submissions.

Assignment briefing:
This assignment will assess your ability to solve a problem using C as a tool. You will be using these C programming elements: repetition (e.g. for.. or while..), selection (if), structures, two dimensional arrays, pointers, I/O and functions. Your task is to write a C program to:

- **Allow the user to input the name and size of a matrix. The contents of the matrix are read in from a file, which has a list of 100 float numbers. (so for eg, if the matrix is 3x2, then the first 2 numbers in the list form the first row, the second 2 numbers in the list form the second row …and so on).
- The matrix of floats must have between 1 and 10 columns and between 1 and 10 rows. The matrix name, size (rows x columns) and contents will be stored in a struct matrix, which is specified below.
- Display the name and contents of the matrix to the screen.
- Calculate the determinant: Ask the user to choose the row and column to form a 2x2 matrix, which is a subset of the original matrix. The 2x2 matrix contents must be stored in a new struct matrix, along with its name and size. The determinant of this matrix is then calculated. (See sample run of program below).
- Display the name and contents of the 2x2 matrix and determinant to the screen.
- Find the inverse of the 2x2 matrix. The inverse matrix contents must be stored in a new struct matrix, along with its name and size.
- Display the name and contents of inverse matrix to the screen.##
- Loop from ** to ##, but allow the user some way of exiting the loop and stopping the program

The program shall define a structure to hold information about the matrices as follows:

```
struct matrix
{
  char name; // to hold the 1 character name of the matrix
   float mValues[10][10]; // to hold the values in the matrix, up to 10 rows x 10 columns
   int nrows;      // to hold the number of rows used in mValues
   int ncols;      // to hold the number of columns used in mValues
};
```

Notes (refer to mark scheme for more details):

1. The array that holds the matrix values has maximum size 10x10, which means that the program must be able to work with any array up to that size. For this program, the minimum array size allowed is 1 x 1.
2. Whenever a new matrix is input or created in the program, the program must allow the user to assign the matrix a name of max 1 character, e.g. 'A', 'B', etc.
3. The program must display to the screen the name and the contents of each matrix in the program, after it has been input or created.
4. The program must allow both square and non-square matrices, e.g. 3 x 3 is square, 3 x 4 is not square.
5. You must **program defensively**. The program must check for out of range values for rows, columns and char name at every possible instance (The program must ask the user to enter again if incorrect!), as well as checking if the input file is available. You also need to take into consideration things like: possibility of generating a 2x2 matrix (because the input matrix is smaller, eg, 2x1), and finding the inverse when the determinant is 0 or does not exist. You should also check that the file exists.
6. **You cannot use global variables. You will get 0 for the whole code. #define can be used.**
7. **You cannot use while(1) loops. You will get 0 for the whole code.**
8. The program must include main( ) and the following 4 functions. The function prototypes must be as follows:

`void matrixInput(struct matrix *mat, FILE *in);` /* to input the size (number of rows and columns) of the matrix, and the name of the matrix, and read in the values from the file (using fscanf) into the 2D array that makes up the matrix */ NOTE, the file opening and checking occurs in main(), eg FILE *fin; fin=fopen(" .txt",'r'); then use fin in the function call.

`void matrixDisplay(struct matrix mat);` /* to display the name of the matrix and the values in the 2D array that makes up the matrix*/

`float matrixDeterminant(struct matrix m1, struct matrix *m2, int *check);` /* to find the determinant of m2, where m2 is a 2x2 subset matrix of m1; returns determinant to main; this function does not display either m1 or m2, name of matrix entered here. If determinant>0, check =1, check=2 if determinant=0, otherwise check is 0 (size of matrix <2x2, this variable is needed for the inverse too */

`void matrixInverse(struct matrix m2, float determinant, struct matrix *m3);` /*to find the inverse (if possible) of the 2x2 matrix; this function does not display either m2 or m3, name of matrix entered here */

9. You are free to add extra functions, for example, for out-of-range checks (for rows and columns), any other defensive programming required, entering name for the matrix.

```
please enter one character name for the matrix, e.g. A, B, etc: A
Enter # rows of the matrix (<10):
5
Enter # columns of the matrix (<10):
7
Number of rows is 5
Matrix A:
Row 0:          7.00    9.00    -8.00   9.00    3.00    -8.00   -5.00
Row 1:          1.00    10.00   10.00   -7.00   10.00   10.00   0.00
Row 2:          6.00    -8.00   -2.00   9.00    6.00    10.00   3.00
Row 3:          -10.00  7.00    9.00    4.00    5.00    5.00    -2.00
Row 4:          3.00    -7.00   4.00    -10.00  -5.00   -10.00  -8.00


Finding the determinant now!
please enter a one character name for the matrix, e.g. A, B, etc: B
Enter row number where to start 2x2 matrix, number needs to be between 0 and 3:
2
Enter column number where to start 2x2 matrix, number needs to be between 0 and 5:
4
The determinant is -20 for
Matrix B:
Row 0:          6.00    10.00
Row 1:          5.00    5.00

Finding the inverse now!
please enter a one character name for the matrix, e.g. A, B, etc: C
The inverse of the matrix is:
Matrix C:
Row 0:          -0.25   0.50
Row 1:          0.25    -0.30

Would you like to input another matrix? (type 1 for yes or 0 for no)
```

Figure 1: A sample screenshot of the expected output from this code.

Marking criteria: The following table lists the components/marking criteria of the assignment and the marks available for each component.

Component	**Marks**
Criteria 1: Have global variables been used? OR Have while(1) loops been used?	Yes/no **If yes, then give 0 for whole program.**
Criteria 2: Program layout and readability including things like how clear is the code and does it have comments, including header comments, meaningful variable names, and indentation?	/1

Criteria 3: Inputting and displaying matrix:
• Does the program use the function prototypes as instructed?
void matrixInput(struct matrix *mat, FILE *in);
void matrixDisplay(struct matrix mat);
• Does this section allow the user to enter number of rows and columns, and reads numbers from the file into appropriate array?
• Does the program display the correct name and the correct contents of each matrix to the screen? Is the display in a sensible format?
	Yes/no
	(if no then 50% off mark in this section)
	
	/1

Criteria 4: Determinant of matrix • Does the program use the function prototypes as instructed? float matrixDeterminant(struct matrix m1, struct matrix *m2, int *check); • Does the program find the correct determinant when original matrix >2x2?	Yes/no (if no then 50% off mark in this section) /1
Criteria 5: Inverse of matrix • Does the program use the function prototypes as instructed? void matrixInverse(struct matrix m, float determinant, struct matrix *m3); • Does the program find the correct inverse when the determinant exist and is >0?	Yes/no (if no then 50% off mark in this section) /1
Criteria 6: Defensive Programming • Does it check for the file? • Does the code guard against incorrect entries at every instance: name, rows & cols?	/1
Criteria 7: Defensive Programming Determinant • Determinant when original matrix < 2x2?	/1
Criteria 8: Defensive Programming Inverse • Inverse for when determinant is 0 or does not exist.	/1
Criteria 9: Details • Looping through the program until user decides to quit. • Clarity of instructions printed to the screen.	/1
Total	/8

Penalties for late submission: 5% reduction in the mark for every day, or part thereof, that the assignment is late, and a mark of zero for submission more than five days late. For more information, see http://www.shef.ac.uk/ssid/exams/policies.

How you should work: **This is an individual assignment**, and it must be wholly your own work. You must not copy algorithms or code for this assignment from any other source. You must not show anyone else your code, nor must you look at anyone else's code, because if you do this is likely to result in similarity in your algorithms and code. You can discuss concepts, but if discussion leads to similar code this will be treated as an unfair means case.

Unfair means: The module leader will submit your C programs to a plagiarism checker. Any suspicions of the use of unfair means will be investigated and may lead to penalties. See http://www.shef.ac.uk/ssid/exams/plagiarism for more information.

Extenuating circumstances: If you have medical or personal circumstances which cause you to be unable to submit this assignment on time or attend your marking appointment, please complete and submit an extenuating circumstances form along with documentary evidence of the circumstances. See http://www.sheffield.ac.uk/ssid/forms/circs for more information.

Help: This assignment briefing, the lectures and labs and the C textbooks on your ACS130 reading list provide all the information that is required to complete this assignment. You need to decide on the algorithm to solve this problem, subject to the instructions in the assignment briefing. You also need to create and debug the C code to solve this problem. However if you need clarifications on the assignment then please **post a question on the ACS130 Blackboard (MOLE) discussion board**. I will not reply to emails unless the query is of a personal nature.