readMe File for my Matrix Calculator C Project

Brief Program Description

My program allows the user to carry out matrix operations such as multiplication, subtraction, addition, transposition, inversion, finding the determinant of a matrix and allows the user to load matrices from file, save matrices to a file, and allows the user to create their own matrix on the terminal.

Commands to run the Program

- First compile the program and create .exe file:
  - gcc -o MatrixCalculator MatrixCalculator.c
- Then run the .exe file:
  - ./MatrixCalculator

How to use the Program

To use the program, first use the commands above to run the program. Once the program is running, a menu of options will appear after which it will ask the user to input their option number. It will then carry out the operation that the user wanted to do, in most cases it will ask the user to enter the values for the matrix that is being created before the selected matrix operation can be carried out.

Once the operation has been carried out, the menu of options will appear again and this loop will keep on going until the user enters the number 10, which will lead to the program quitting.

Further Explanation of Program

int Show_Menu() Function

I have created a function called Show_Menu(), which displays the Menu of the Matrix Program every time it is called. It allows the user to enter a number of the option they pick and then returns it.

void option_repeat() Function

This function will loop through the Show_Menu() and the doOption() function until the user enters 10 to quit the program.

void doOption(int option) Function

This function will take the user-selected option in as an argument and then using the switch case statements, will carry out the appropriate task.

void matrix_load_from_file(int ** firstMatrix, int ** secondMatrix)Function

This function will first initialise an empty File pointer and 2 empty double pointers (just with memory allocated to them, but with no values in them) and then will open the data file, “MatricesStore.txt”, using the file pointer in read mode. Then by using fscanf, it will look for integers and then store them in the relevant position in the matrix. We pass two empty double pointers (sample and sample1) to allow us to return more than one variable from the function. At the end of this function, we assign sample and sample1 to the two arrays that we have just loaded from the file respectively.

void matrix_save_to_file(int ** matrix) Function

This function will initialise and declare a File pointer to open the data file, “MatricesStore.txt” in a+ mode, which means that its open for reading and appending. It will then write the double pointer/matrix passed into the function as an argument into the file using fprintf.

void matrix_add() Function

This function will initialise two new double pointers fully, by assigning values into them using the initialise_matrix() function. It will then create a third double pointer which will store the result of our addition. Using malloc, we first dynamically allocate memory to the double pointer as a whole and then using a for loop, we dynamically allocate memory to each row of the double pointer. We carry out the addition in a nested for loop. After the printing has been done, we free the memory already occupied by the double pointer, so that it can be used elsewhere.

void matrix_subtract() Function

This function is almost identical in nature to the matrix_add() function, only difference is that here subtraction is taking place, as opposed to the addition in the matrix_add() function.

void matrix_transpose() Function

This function will initialise a matrix by calling the initialise_matrix() function, save that into the double pointer called first. Then we will initialise an empty double pointer using the initialise_matrix_without_input() function in which we will store our transposed matrix.

int ** initialise_matrix(int* sample, int* sample2) Function

This function will initialise a full double pointer. Using malloc, we first dynamically allocate memory to the double pointer as a whole and then using a for loop, we dynamically allocate memory to each row of the double pointer. Then we ask the user to enter his/her values for the matrix and the program will read this input using scanf and then store in the relevant position in the matrix.

int ** initalise_matrix(int* number_of_rows, int* number_of_columns)Function

This function will initialise a full double pointer. Using malloc, we first dynamically allocate memory to the double pointer as a whole and then using a for loop, we dynamically allocate memory to each row of the double pointer. We will leave this double pointer empty however, because it is to be called by the matrix_load_from_file() function so that it can store values into the matrix in that function.

void matrix_multiply() Function

This function will first initialise 2 new double pointers by calling the initialise_matrix() function and then initialise the third double pointer using the initialise_matrix_without_input() function which will be the matrix in which we will be storing the result of our matrix multiplication. Then we will multiply the matrices. Once this has been calculated, we assign this sum into the result matrix and print that out at the end of the function.

void matrix_determinant() Function

This function will first initialise a new double pointer by calling the initialise_matrix() function and then work out the determinant of this matrix and print it out on the terminal.

void matrix_mulitply_by_scalar() Function

This function will first initialise a new double pointer by calling the initialise_matrix() function and then an empty double pointer using initalise_matrix_without_input() function, into which we will store the result of the scalar multiplication.

void matrix_inversion() Function

We calculate the determinant of the input matrix and save that into a variable called determinant. We then do a maths calculation in which the value of the determinant is used and then we put the inverted value into the empty double pointer. After this is done, we print out the inverted matrix on the terminal.