Description

Main Goal:

This C program *matrix.c* is focusing on matrix basic operations including:

1. Addition of two matrices
2. Subtraction of two matrices
3. Multiplication of two matrices
4. Multiplication of a matrix by a scalar
5. Check if two matrices are equal to each other
6. Calculate the trace of a matrix
7. Calculate the determinant of a matrix
8. Take the power of a matrix

Environment:

To implement these projects, this program contains only C standard libraries, including:

<inttypes.h>, <stdint.h>, <stdio.h>, <string.h>, <stdlib.h>, <math.h>

Matrix file format:

Each element in the matrix is in the data type of double-precision floating-point number.

All the input and output matrices are in .txt file format. In each row, whitespace is used for separating each element.

Ex: A 4x4 matrix

first.txt

```
2.2 2.1 3.0 4.5
-14 5.1 -6.0 4.2
7.1 -8.0 9.2 4.1
4.2 4.1 -4.0 5.2
```

How to run the program:

The user needs to use commend line in terminal for running this program. **Make sure the current path is on the directory where the source files are.**

For powershell: type ".\" before the program name, followed by other commend line arguments separated by whitespace. The program name doesn't need its extension, the file name needs its extension (.txt).

Ex: .\matrix add addM1.txt addM2.txt addResult.txt

For cwd: type the program name, followed by other commend line arguments separated by whitespace. The program name doesn't need its extension, the file name needs its extension (.txt).

Ex: matrix add addM1.txt addM2.txt addResult.txt

Handle General Exceptions:

- If there is no commend line argument input, it will print an error message: "You need to put some arguments in command line.".
- If there is a typo or not giving a function name in commend line, it will print an error message: "Please use one of the available functions.".
- If there is a problem when opening a file, it will print an error message "Cannot open file".
- If there is a problem when reading or writing a file, it will print an error message specifying which file has the problem.
- If there is a problem when allocating memory for dynamic arrays, it will print an error message specify which file has the problem "Failed to allocate memory for &filename".
- For any matrix that is not valid (does not have same number of elements in each row, or has non-numerical elements), the program will stop and print an error message specify which file and which row the problem has occurred.

Functions:

```
void countRowCol(const char *filename, uint64_t *row, uint64_t *col)
```

Description:

Read matrix from file and find its row and col then pass them to some variable.

Use fgets and sscanf to scan each line of the file, and count rows and elements in each row (column).

Use pointer to pass the row and col that we find from this function for future use.

Parameters:

filename is a char array which is the name of file we want to open

row is a pointer to row value of the fixed-size integer

col is a pointer to column value of the fixed-size integer

```
int verifyMatrix(const char *filename)
```

Description:

Read matrix from file and verify if the matrix in the file is valid.

Verify whether the matrix has same amount of numbers in each row, and whether the matrix contain some non-numerical items.

Parameters:

filename is a char array which is the name of file we want to open

Return:

An integer, if it's 1, then the matrix is invalid and will report the error message. If it's 0, then this matrix is valid.

```
void add(const char *filename1, const char *filename2, const char *storeFile)
```

Description:

Read matrices from file 1 and file 2 and add them up.

Store matrices in dynamic arrays. If matrices are in the same dimension, add them up and store the result in a new dynamic array.

Parameters:

filename1 is a char array which is the name of the first matrix file
filename2 is a char array which is the name of the second matrix file
storeFile is a char array which is the name of the file we store the result

Example:

addM1.txt:

2.2 2.1 3.0 4.5
-14 5.1 -6.0 4.2
7.1 -8.0 9.2 4.1
4.2 4.1 -4.0 5.2

addM2.txt:

1.2 2.0 3.0 -2.0
2.34 5.2 11.0 -3.123
6.4 -2.0 2.22 7.1
1.34 -7.0 2.1 5.2

Use addM1.txt and addM2.txt, run in powershell:

```
.\matrix add addM1.txt addM2.txt addResult.txt
```

The first argument is program name “matrix”, the second argument is the function name, the third and fourth argument are two matrices files’ name, and the fifth argument is the file we store the result.

The output will generate a .txt file addResult.txt, with matrix:

3.400000 4.100000 6.000000 2.500000
-11.660000 10.300000 5.000000 1.077000
13.500000 -10.000000 11.420000 11.200000
5.540000 -2.900000 -1.900000 10.400000

Exceptions handle:

- If two matrices are having different dimension,
  `.\matrix add testM3x4.txt testM4x3.txt addResult.txt`
  It will print an error message: “Two matrices don't have the same dimension, cannot be added together.”, and will not generate the file “addResult.txt”.
- If there are missing arguments,

  `.\matrix add testM3x4.txt addResult.txt`

  It will print an error message: “The number of arguments is incorrect.” and will not generate the file “addResult.txt”.

```
void subtract(const char *filename1, const char *filename2, const char *storeFile)
```

Description:

Read matrices from file 1 and file 2 and use first matrix subtract the second one.

Store matrices in dynamic arrays. If matrices are in the same dimension, subtract each other and store the result in a new dynamic array.

Parameters:

filename1 is a char array which is the name of the first matrix file
filename2 is a char array which is the name of the second matrix file
storeFile is a char array which is the name of the file we store the result

Example:

subM1.txt:

2.2 2.1 3.0 4.5
-14 5.1 -6.0 4.2
7.1 -8.0 9.2 4.1
4.2 4.1 -4.0 5.2

subM2.txt:

1.2 2.0 3.0 -2.0
2.34 5.2 11.0 -3.123
6.4 -2.0 2.22 7.1
1.34 -7.0 2.1 5.2

Use subM1.txt and subM2.txt, run in powershell:

```
.\matrix subtract subM1.txt subM2.txt subResult.txt
```

The first argument is program name “matrix”, the second argument is the function name, the third and fourth argument are two matrices files’ name, and the fifth argument is the file we store the result.

The output will generate a .txt file subResult.txt, with matrix:

1.000000 0.100000 0.000000 6.500000
-16.340000 -0.100000 -17.000000 7.323000
0.700000 -6.000000 6.980000 -3.000000
2.860000 11.100000 -6.100000 0.000000

Exceptions handle:

- If two matrices are having different dimension,
  `.\matrix subtract testM3x4.txt testM4x3.txt subResult.txt`
  It will print an error message: “Two matrices don't have the same dimension, cannot be subtracted together.” and will not generate the file “subResult.txt”.
- If there are missing arguments,
  `.\matrix subtract testM3x4.txt subResult.txt`
  It will print an error message: “The number of arguments is incorrect.” and will not generate the file “subResult.txt”.

```
void multiply(const char *filename1, const char *filename2, const char *storeFile)
```

Description:

Read matrices from file 1 and file 2 and multiply them up.

Store matrices in dynamic arrays. If matrix one's column number is same as matrix two's row number, multiply them up and store the result in a new dynamic array.

Matrix multiplication only appliable if the first matrix's column is equal to the second matrix's row. The result is a m x n matrix, where the m is the row of the first matrix, n is the column of the second matrix. Each element in the result is the dot product between first matrix and second matrix with their corresponding row and column. For example, the first number in the result is equal to the dot product of the first row in first matrix with the first column in the second matrix.

Parameters:

filename1 is a char array which is the name of the first matrix file
filename2 is a char array which is the name of the second matrix file
storeFile is a char array which is the name of the file we store the result

Example:

mulM1.txt:

```
2.2 2.1 3.0 4.5
-14 5.1 -6.0 4.2
7.1 -8.0 9.2 4.1
```

mulM2.txt:

```
1.2 2.0 3.0
2.34 5.2 11.0
6.4 -2.0 2.22
1.34 -7.0 2.1
```

Use mulM1.txt and mulM2.txt, run in powershell:

```
.\matrix multiply mulM1.txt mulM2.txt mulResult.txt
```

The first argument is program name "matrix", the second argument is the function name, the third and fourth argument are two matrices files' name, and the fifth argument is the file we store the result.

The output will generate a .txt file mulResult.txt, with matrix:

```
32.784000 -22.180000 45.810000
-37.638000 -18.880000 9.600000
54.174000 -74.500000 -37.666000
```

Exceptions handle:

- If two matrices are having different dimension,
  `.\matrix multiply testM4x3.txt mulM2.txt mulResult.txt`
  It will print an error message: “Two matrices don't have the right dimension for doing multiplication.”, and will not generate the file “mulResult.txt”.
- If there are missing arguments,
  `.\matrix multiply testM3x4.txt mulResult.txt`
  It will print an error message: “The number of arguments is incorrect.” and will not generate the file “mulResult.txt”.

```
void multiply_scalar(const char *filename, char *scalar, const char *storeFile)
```

Description:

Read matrix from file and multiply it by scalar.

Store matrix in dynamic arrays. Multiply each element of matrix by the scalar number and store the result in a new dynamic array.

Matrix multiplication by scalar is simply multiply each elements in the matrix by a given scalar number.

Parameters:

filename is a char array which is the name of the first matrix file.
scalar is a char array which is a double number in string form.
storeFile is a char array which is the name of the file we store the result

Example:
mul_scalarM.txt:

1.1 2.2 3.3
0.0 -2.0 1
2.5 3.5 4.5

Use mul_scalarM.txt and randomly select a scalar, assume it is 5 there, run in powershell:

```
.\matrix multiply_scalar mul_scalarM.txt 5 mul_scalarResult.txt
```

The first argument is program name “matrix”, the second argument is the function name, the third argument is the matrix file’s name, the fourth argument is the scalar number, and the fifth argument is the file we store the result.

The output will generate a .txt file mul_scalarResult.txt, with matrix:

5.500000 11.000000 16.500000
0.000000 -10.000000 5.000000
12.500000 17.500000 22.500000

Exceptions handle:

- If the scalar is not a valid number,
  `.\matrix multiply_scalar testM4x3.txt scalar mul_scalarResult.txt`
  It will print an error message: “Please use a valid number for the scalar.” and will not generate the file “mul_scalarResult.txt”.
- If there are missing arguments,
  `.\matrix multiply_scalar testM3x4.txt mul_scalarResult.txt`
  It will print an error message: “The number of arguments is incorrect.” and will not generate the file “mul_scalarResult.txt”.

```
void is_equal(const char *filename1, const char *filename2)
```

Description:

Read matrices from file 1 and file 2 and test whether they are equal to each other.

Store matrices in dynamic arrays. If matrices are in the same dimension, compare their elements one by one. Print out the message if or not these two matrices are equal.

Parameters:

filename1 is a char array which is the name of the first matrix file
filename2 is a char array which is the name of the second matrix file

Example:

equalM1.txt:

```
1.1 2.2 3.3
0.0 -2.0 1
2.5 3.5 4.5
```

equalM2.txt:

```
1.1 2.2 3.3
0.0 -2.0 1
2.5 3.5 4.5
```

not_equalM.txt:

```
1.1 2.2 3.3
0.0 -2.0 1
2.5 3.5 4.2
```

Use equalM1.txt and equalM2.txt to test equality, run in powershell:

```
.\matrix is_equal equalM1.txt equalM2.txt
```

The first argument is program name “matrix”, the second argument is the function name, the third and fourth argument are two matrix files’ names.

It will print a message in the terminal: “True, two matrices are equal to each other.”

Use equalM1.txt and not_equalM.txt to test inequality, run in powershell:

```
.\matrix is_equal equalM1.txt not_equalM.txt
```

The first argument is program name “matrix”, the second argument is the function name, the third and fourth argument are two matrix files’ names.

It will print a message in the terminal: “False, two matrices are not equal to each other.”

Exceptions handle:

- If two matrices are having different dimension,
  `.\matrix is_equal testM4x3.txt testM3x4.txt`
  It will print an error message: “Two matrices don't have the same dimension, cannot compare equality together.”
- If there are missing arguments,
  `.\matrix is_equal testM3x4.txt`
  It will print an error message: “The number of arguments is incorrect.

```
void trace(const char *filename)
```

Description:

Read matrices from file and compute its trace.

Store matrix in dynamic arrays. If the matrix is a square matrix, add all the diagonal values. Print the result in the terminal.

Parameters:

filename is a char array which is the name of the first matrix file.

Example:
traceM.txt:

```
1.1 2.2 3.3
0.0 -2.0 1
2.5 3.5 4.5
```

Use traceM.txt, run in powershell:

```
.\matrix trace traceM.txt
```

The first argument is program name “matrix”, the second argument is the function name, the third argument is the matrix file’s name

It will print a message in the terminal: “The trace of this matrix is 3.600000.”

Exceptions handle:

- If the matrix is not a square matrix,
  `.\matrix trace testM4x3.txt`
  It will print an error message: “The matrix is not a square matrix, cannot compute its trace.”
- If there are missing arguments,
  `.\matrix trace`
  It will print an error message: “The number of arguments is incorrect.”

```
double determinant(double *m, uint64_t n)
```

Description:

Given a nxn matrix and its size n, compute the determinant

Use recursion for nxn matrix with n>2. The base cases are when n=1 and n=2.

When n = 1. The determinant is the matrix itself.

When n = 2. The determinant of matrix [a1,a2;a3,a4] is a1*a4-a2*a3.

When n > 2, the determinant is computed by: choose a row, use the numbers in that row as the factor, multiply with the determinant of a minor matrix, where the minor matrix size is (n-1)x(n-1). The minor matrix is constructed by the numbers which are not in the same row and column with the factor. Then multiply by (-1)^(a+b), where a is the row of the factor, b is the column of the factor. Finally we add all the terms up, get the matrix's determinant.

Ex: Assume a 3x3 matrix

1 2 3 4 5 6 7 8 9

- Choose the first row for factors. The first factor is 1. The minor matrix is [5,6;8,9]. The first term is 1*(5*9-6*8)*(-1)^(1+1) = -3
- The second factor is 2. The minor matrix is [4,6;7,9]. The second term is 2*(4*9-6*7)*(-1)^(1+2) = 12
- The third factor is 3. The minor matrix is [4,5;7,8]. The third term is 3*(4*8-5*7)*(-1)^(1+3) = 9
- The determinant of this matrix is -3+12+9 = 0

Parameters:

m is a double array contains the nxn matrix
n is a fixed-size integer, it is the size of nxn matrix m.

Return:

A double value, that is the determinant of parameter matrix m.

```
void det(const char *filename)
```

Description:

Read matrices from file and compute its determinant by function determinant.

Store matrix in dynamic array. If matrix is a square matrix, run function determinant to compute determinant. Print the result in terminal.

Parameter:

filename is a char array which is the name of the matrix file.

Example:
detM.txt:

2.2 2.1 3.0 4.5
-14 5.1 -6.0 4.2
7.1 -8.0 9.2 4.1
4.2 4.1 -4.0 5.2

Use detM.txt, run in powershell:

```
.\matrix det detM.txt
```

The first argument is program name “matrix”, the second argument is the function name, the third argument is the matrix file’s name

It will print a message in the terminal: “The determinant is 4864.834800.”

Exceptions handle:

- If the matrix is not a square matrix,
  ```
  .\matrix det testM4x3.txt
  ```
  It will print an error message: “The matrix is not a square matrix, cannot compute its determinant.”
- If there are missing arguments,
  ```
  .\matrix det
  ```
  It will print an error message: “The number of arguments is incorrect.”

```
void power(const char *filename, char *factor, const char *storeFile)
```

Description:

Read matrices from file and compute its power by a given exponent

Store matrices in dynamic arrays. If matrix is a square matrix, multiply itself multiple times. Store the result in a file.

Parameter:

filename is a char array which is the name of the matrix file.

factor is a char array. It's the exponent, an integer in string form.

storeFile is a char array which is the name of the file we store the result.

Example:
powM.txt:

2.2 2.1 3.0
-14 5.1 -6.0
7.1 -8.0 9.2

Use powM.txt, and randomly select a exponent, assume it is 3 there, run in powershell:

```
.\matrix power powM.txt 3 powResult.txt
```

The first argument is program name “matrix”, the second argument is the function name, the third argument is the matrix file’s name, the fourth argument is the exponent number, and the fifth argument is the file we store the result.

The output will generate a .txt file powResult.txt, with matrix:

267.568000 -223.863000 240.960000
-1850.480000 945.831000 -1877.820000
2910.302000 -1333.745000 2592.008000

Exceptions handle:

- If the matrix is not a square matrix,
  `.\matrix power testM4x3.txt 3 powResult.txt`
  It will print an error message: “The matrix is not a square matrix, cannot compute its power.” and will not generate the file “powResult.txt”.
- If the exponent is not a valid number,
  `.\matrix power testM4x3.txt exponent mul_scalarResult.txt`
  It will print an error message: “Please use a valid integer for the exponent.” and will not generate the file “powResult.txt”.
- If there are missing arguments,
  `.\matrix power testM4x3.txt powResult.txt`
  It will print an error message: “The number of arguments is incorrect.” and will not generate the file “powResult.txt”.

```
int main(int argc, char *argv[])
```

Description:

Main function, use commend line argument run every functions from above.

Parameter:

argc, an integer which counts the number of arguments passed to the program

argv[], an array of argc pointers to strings, contain the actual arguments passed.

Return:

Return 0 by default, return 1 if some exception happens.