Problem 1: (6 Points)

Your task is to read two 3D matrices from binary files (matrix1.bin, matrix2.bin), perform element-wise multiplication, and then write the result to another binary file (result.bin). You can use the program attached create_matrix.c (Download create_matrix.c and generate two binary files. The code generates matrices and writes them to binary files. It prompts for a filename, creates a binary file with that name, and then prompts for the size (n) of the square matrix. It then generates random numbers for the elements of the matrix and writes them to the file in row-major order.) To generate two matrices (matrix1.bin and matrix2.bin), you can run this code two times, each time providing a different filename for each matrix.

Assume the binary files contain cubic matrices of integers (i.e., the number of rows is equal to the number of columns and depth). Each binary file starts with a single integer, n, that specifies the dimension of the matrix, i.e., n x n x n. After the dimension, the file has n^3 integers indicating the elements of the matrix in a row-major order. Each integer is a 4B int type value. Row-major order is a method of storing multidimensional matrices in linear storage like memory. In a 3-dimensional matrix, row-major order means that the elements of the matrix are stored plane by plane, row by row.

Here is the detailed process: Read the first integer in each file to find the dimension (n) of the matrix. If the matrices are not compatible for multiplication (i.e., two different n values are read from two matrix files), or if n is greater than 100, print out an error message and terminate the execution. After knowing the dimension, read the next n^3 integers from each file as the elements of the two matrices. Use matrix indexing to store these elements in two 3D matrices (matrix1 and matrix2). Perform element-wise multiplication on matrix1 and matrix2 to produce a new matrix (result_matrix). Write the result_matrix to a new binary file. The file should start with the matrix dimension (n) followed by all n^3 elements of the result_matrix in a row-major order.

Example Code Structure:

```
#include <stdio.h>
int main() {
int n;
int matrix_1[n][n][n], matrix_2[n][n][n], result_matrix[n][n][n];
        // Open matrix1.bin and matrix2.bin for reading
        // Read the dimension of the matrices
        // Read the elements of the matrices
        // Close the input files
        // Perform element-wise multiplication
        // Open result.bin for writing
        // Write the dimension of the result_matrix
        // Write the elements of the result_matrix
```

```
    // Close the output file
    return 0;
}
```

To test your program, you may write another program, which generates two 3D matrices, one containing random numbers, the other being a 3D identity matrix, and writes these two matrices into the two files. A 3D identity matrix is a cubic matrix where all the elements are zero except for the elements on the main diagonal (i.e., where the row, column, and depth indices are all the same), which are all one.

2. Problem 2 (4 Points)

Write a C program that traverses a directory and its subdirectories recursively,counting the total number of lines across all text files (.txt) present. The Program should print the total line count when it finishes. For simplicity, assume that all files you need to consider are those with the .txt extension, and those files are regular files. So you do not need to handle symbolic links. You can use scanf() to read the pathname of the directory into a string. The max length of any pathnames is 1000.