Part 2b Documentation

Program Description

The program reads the output files of part one as inputs and writes the matrix multiplication of each unique two matrix permutation into output files. Since matrix multiplication is asymmetric, only 25(the permutations of a set of 5, and a subset of 2, with repetition) files are generated.

Important Library Details

- Eigen
    - Library path: the headers for the Eigen library are located in /usr/include/eigen3 on my Linux machine.
    - Library version: I have installed Eigen version 3.4.0.

Marginal Cases

- Invalid inputs:
    - If, for some reason, the outputs of part one have not been generated, an assertion in ReadMatFile ensures the program will end harmlessly.
    - The outputs of part one have already been verified, therefore it is not problematic to assume all inputs are two-dimensional double matrices with correctly labeled dimensions.
    - The WriteMatProductFile wrapper methods for the MatProducts methods check for the dimensions of input arrays and perform matrix multiplication only if the dimensions are identical.
- Invalid computations:
    - All important computations in the Eigen implementation methods were handled by Eigen, and the outputs have been checked.
    - For custom implementation methods, outputs have also been checked.

Design Choices

- The MatProduct method was split into two implementations:
    - MatProductEigen which uses the built in Eigen matrix multiplication functionality. I made this implementation for the sake of familiarizing myself with the Eigen library.
    - MatProductCustom which implements a custom matrix multiplication algorithm. I made this implementation to demonstrate that I understand how matrix by matrix multiplication works.
- The WriteMatProductFile wrapper methods for the MatProduct methods were created to integrate error handling and matrix multiplication into single methods.

- The choice of words in the assignment “name_p2b_out12.txt”, “name_p2b_out13.txt”, … etc. as opposed to “name_p2b_out11.txt”, “name_p2b_out12.txt”, … etc. seems to imply that matrices multiplied by themselves might not need to be generated. However, just to be safe, they will be generated anyway.
- Visualizing the permutations of a set of 5 and a subset of 2 with repetition gives us: aa ab ac ad ae ba bb bc bd be ca cb cc cd ce da db dc dd de ea eb ec ed ee. Noticing the fact that the second character of each permutation proceeds in alphabetical order just like the first character, we can create each permutation with a simple series of for loops.

Pseudocode

```
// Multiplies two matrices in a custom implementation and returns the product. Assumes input
// matrices can be multiplied.
Matrix MatProductCustom(Matrix input_1, Matrix input_2)

// Multiplies two matrices using Eigen and returns the product. Assumes input matrices
// can be multiplied.
Matrix MatProductEigen(Matrix input_1, Matrix input_2)

// Read the matrix at file_path’s data, create a matrix object with that data, and return the matrix
// object.
Matrix ReadMatFile(string read_file_path)

// Write a matrix mat’s dimensions and data to a file at file_path.
Void WriteMatFile(Matrix mat, string file_path)

// Write the matrix product of the two input matrices, or an error message, to a file at
// output_path using MatProductCustom.
Void WriteMatProductFileCustom(Matrix input_1, Matrix input_2, string output_filepath)

// Write the matrix prduct of the two input matrices, or an error message, to a file at output_path
// using MatProductEigen.
Void WriteMatProductFileEigen(Matrix input_1, Matrix input_2, string output_filepath)

Int main():
        Const string kMat1Path = “../part_one/jhartt_p1_mat1.txt”
        Const string kMat2Path = “../part_one/jhartt_p2_mat1.txt”
        Const string kMat3Path = “../part_one/jhartt_p3_mat1.txt”
        Const string kMat4Path = “../part_one/jhartt_p4_mat1.txt”
        Const string kMat5Path = “../part_one/jhartt_p5_mat1.txt”
        Const Matrix kMat1 = ReadMatFile(kMat1Path)
        Const Matrix kMat2 = ReadMatFile(kMat2Path)
        Const Matrix kMat3 = ReadMatFile(kMat3Path)
        Const Matrix kMat4 = ReadMatFile(kMat4Path)
```

```
    Const Matrix kMat5 = ReadMatFile(kMat5Path)
    Const Vector<Matrix> kMatArr = {kMat1, kMat2, … kMat5}

    // The permutations of a set of 5 and a subset of 2 with repetition is implemented
    // somewhat simply here
    For (int first_mat_num = 0; first_mat_num < 5; first_mat_num++) {
        For (int second_mat_num = 0; second_mat_num < 5; second_mat_num++) {
            String output_path = “jhartt_p2b_out” + toString(first_mat_num + 1) +
                                 toString(second_mat_num + 1) + “.txt”;

            // Alternate MatProduct implementations for demonstration purposes
            If ((first_mat_num + second_mat_num) % 2 == 0) {
                WriteMatProductFileCustom(kMatArr[first_mat_num],
                                 kMatArr[second_mat_num], output_path)
            } else {
                WriteMatProductFileEigen(kMatArr[first_mat_num],
                                 kMatArr[second_mat_num], output_path)
            }
        }
    }

    Return 0

Matrix MatProductCustom(Matrix input_1, Matrix input_2):
    Matrix out_mat(input_1.rows(), input_2.cols())

    Iterate through out_mat row indices:
        Iterate through out_mat column indices:
            Int element_sum = 0

            For (int inner_index = 0; inner_index < input_1.cols(); inner_index++):
                Element_sum += input_1(row, inner_index) *
                                        input_2(inner_index, col)

            out_mat(row, col) = element_sum

    Return out_mat
```