Part 3 Documentation

Program Description

The program generates a custom input file, which it then uses to perform linear algebra calculations on and puts the output into a file. Then, the program runs calculations on the class-provided input and stores those in a separate output file. The class input file is filled with invalid input data, so input validation will be needed.

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.
    - If the operation is invalid, an error will be printed
    - If the number of vector components is invalid, an error will be printed
- 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 methods to handle each method have been implemented with Eigen where possible, and custom implementations otherwise, with dot product as an exception. The dot product will be implemented custom to demonstrate my understanding of the dot product.
- Implementation table(C for custom, E for Eigen, CE for both):

Method	Implementation
Addition(AD)	E
Subtraction(SU)	E
Scaling(SC)	C
Dot Product(DO)	C

Cosine Angle(CO)	C
Projection(PR)	C

- In the interest of my sanity, I will use exceptions and a try-catch statement to simplify error detection, even if it violates the Google C++ Style Guide.
- I am using an enum for operations to make the code more human readable.

Pseudocode

```
Enum class Operation {
        kAddition = 0,
        kSubtraction = 1,
        kScaling = 2,
        kDotProduct = 3,
        kCosineAngle = 4,
        kProjection = 5
}

// Provides a structured input object
Struct calculation {
        Operation operation
        Eigen::Vector vector_1
        Eigen::Vector vector_2
}

// Converts a raw calculation (array of strings) to a calculation.  Will fail if the operation is invalid,
// the number of vector components are invalid, or the vector components are not doubles.
Calculation ConvertToCalculation(std::vector<string> raw_calculation)

// Generates an input file.
Void GenInputFile(string file_path)

// Read the  at file_path's data, create a jagged 2D string array with that data, and return the 2d
// array object.  Each row will contain each word in the input file's given line.
std::Vector<std::vector<string>> ReadInputFile(string input_file_path)

// Splits a string at the spaces.
std::vector<string> SplitString(string str)

// Gets the dot product of two vectors in a custom implementation and returns the product.
Double VectorDotProductCustom(Eigen::Vector input_1, Eigen::Vector input_2)
```

```
// Calculates the angle between input_1 and input_2 using the theorem a dot b = ||a|| ||b||
// cos(theta).
double VectorCosineAngle(Eigen::Vector input_1, Eigen::Vector input_2)

// Calculates the orthogonal projection of input_2 onto input_1.
Eigen::Vector VectorProjection(Eigen::Vector input_1, Eigen::Vector input_2)

// Calculates input_1 scaled by the magnitude of input_2.
Eigen::Vector VectorScaling(Eigen::Vector input_1, Eigen::Vector input_2)

swit
Void WriteVectorCalculationsFile(string input_file_path, string output_file_path)

Int main():
        GenInputFile(“jhartt_p3_input.txt”);

        WriteVectorCalculationsFile(“jhartt_p3_input.txt”, “jhartt_p3_output.txt”);
        WriteVectorCalculationsFile(“class_p3_input.txt”, “class_p3_output.txt”);

        Return 0

Calculation ConvertToCalculation(std::vector<string> raw_calculation):
        assert(raw_calculation.size = 5)

        String raw_operation = raw_calculation[0]

        Calculation calculation

        If (raw_operation = “AD”):
                Calculation.operation = Operation::kAddition
        Else if (raw_operation = “SU”):
                Calculation.operation = Operation::kSubtraction
        Else if (raw_operation = “SC):
                Calculation.operation = Operation::kScaling
        Else if (raw_operation = “DO”):
                Calculation.operation = Operation::kDotProduct
        Else if (raw_operation = “CO”):
                Calculation.operation = Operation::kCosineAngle
        Else if (raw_operation = “PR”):
                Calculation.operation = Operation::kProjection
        Else:
                throw runtime_error("invalid operation")

        // If an stod below fails, one of the raw components isn’t a double
```

```
        Calculation.vector_1 = Eigen::Vector(stod(raw_calculation[1]), stod(raw_calculation[2]))
        Calculation.vector_2 = Eigen::Vector(stod(raw_calculation[3]), stod(raw_calculation[4]))

        Return calculation

Void GenInputFile(string file_path):
        Open gen_file at file_path

        Set up random number generator

        Gen_file << 'AD ' << random component << ' ' << random component << ' '
                        << random component << ' ' << random component << '\n'
        Gen_file << 'SU ' << random component << ' ' << random component << ' '
                        << random component << ' ' << random component << '\n'
        Gen_file << 'SC ' << random component << ' ' << random component << ' '
                        << random component << ' ' << random component << '\n'
        Gen_file << 'DO ' << random component << ' ' << random component << ' '
                        << random component << ' ' << random component << '\n'
        Gen_file << 'CO ' << random component << ' ' << random component << ' '
                        << random component << ' ' << random component << '\n'
        Gen_file << 'PR ' << random component << ' ' << random component << ' '
                        << random component << ' ' << random component << '\n'
        Gen_file << 'XD ' << random component << ' ' << random component << ' '
                        << random component << ' ' << random component << '\n'
        Gen_file << 'SU ' << random component << ' ' << random component << ' '
                        << random component << '\n'
        Gen_file << 'PR ' << 'C' << ' ' << random component << ' '
                        << random component << ' ' << random component << '\n'

        Close gen_file

std::vector<string> SplitString(string str):
        Stringstream string_stream(str);
        std::vector<string> out;

        Until stream not good:
                String temp_str
                String stream >> temp str
                out.append(temp str)

        Return out

std::Vector<std::vector<string>> ReadInputFile(string input_file_path):
        std::vector<std::vector<string>> out;
```

```
    Open file stream

    Until file stream not good:
        string line;
        Line = getline
        Std::vector<string> raw_calculations = splitString(line)
        out.append(raw_calculations)

    Return out

Void WriteVectorCalculationsFile(string input_file_path, string output_file_path):
    std::Vector<std::vector<string>> raw_calculations = ReadInputFile(input_file_path)

    Open output file at output_file_path

    For each std::vector<string> raw calculation in raw calculations:
        Try:
            Calculation calculation = ConvertToCalculation(raw calculation)

            Switch calculation.operation:
                Per case, call the corresponding Eigen or Custom method and
                        write the result to the file
                Per default, write “You found a unicorn!\n” to the file
        Catch:
            Output file << “Error: improper input ” << endl

    Close output file
```