Assignment: Operator Overloading

CSE 108:OOP Sessional

Problem Statement

You are tasked with implementing three C++ classes:

1. `Fraction`
2. `FractionVector`
3. `FractionMatrix`

Each of these classes should model mathematical operations commonly performed on fractions, vectors, and matrices.

Task Breakdown

Task 1: Fraction Class (20 Marks)

Implement a class `Fraction` with the following specifications:

- Attributes: `numerator` and `denominator`.
- Ensure fractions are always stored in simplified form (reduce to lowest terms after every operation).
- The denominator must never be zero. Handle invalid input appropriately.
- Overload the arithmetic operators: `+`, `-`, `*`, `/` to support:
  - Operations between two `Fraction` objects.
  - Operations between a `Fraction` and a `float` value.
  - Overloading must ensure expressions like `Fraction + float` and `float + Fraction` both work correctly. Similarly, for subtraction, multiplication, and division.
- Overload compound assignment operators: `+=`, `-=`, `*=`, `/=` for operations with another `Fraction` or a `float`.
- Overload the stream insertion operator `<<` to display a `Fraction` in the form of `numerator/denominator`.

Task 2: FractionVector Class (30 Marks)

Implement a class `FractionVector` with the following specifications:

- Internally store the list of fractions using a dynamically allocated array of `Fraction` objects (i.e., using raw pointers and dynamic memory allocation).
- Overload the subscript operator `[]` to allow both reading and writing individual `Fraction` elements by index.
  - Example: `vec[2]` should return a reference to the 3rd element, allowing both assignment and retrieval.
- Overload vector addition and subtraction operators: `+`, `-`, to perform element-wise operations between two `FractionVector` objects.
  - Ensure both vectors are of the same size before performing operations.
- Overload scalar multiplication and division operators to allow multiplying or dividing a `FractionVector` by a single `Fraction` scalar on either side.
  - Support expressions like `vec * frac`, `frac * vec`, and `vec / frac`.
- Overload the `*` operator to compute the **dot product** of two `FractionVector` objects.
  - The dot product of two vectors $A = [a_1, a_2, ..., a_n]$ and $B = [b_1, b_2, ..., b_n]$ is defined as $a_1 \cdot b_1 + a_2 \cdot b_2 + ... + a_n \cdot b_n$.
  - Return the result as a `Fraction`.
  - Check and enforce that both vectors have the same length before computing the dot product.
- Implement a method `value()` that computes the **magnitude (L2 norm)** of the vector:

$$\texttt{value()} = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$$

- Overload the stream insertion operator `<<` to print all the elements of the vector in a readable format.

Note: You must reuse the `Fraction` class from Task 1 for all internal operations and storage.

Task 3: FractionMatrix Class (40 Marks)

Implement a class `FractionMatrix` with the following specifications:

- Internally store the matrix using:
  - An array of `FractionVector` objects representing the rows.
  - An array of `FractionVector` objects representing the columns.

  - Use dynamic memory allocation (raw pointers) for both arrays.

- Overload the subscript operator `[]` to access rows:
  - `matrix[i]` should return a reference to the `FractionVector` object representing the i-th row.
  - This allows element access via expressions like `matrix[i][j]`, where `matrix[i]` returns the i-th row as a `FractionVector` and `[j]` accesses its j-th element.

- Implement a method `getColumn(int index)` that:
  - Returns the `FractionVector` corresponding to the specified column index.
  - Provides safe access (with bounds checking) to column vectors.

- Overload matrix addition and subtraction operators: `+`, `-`.
  - Perform element-wise addition or subtraction of matrices.
  - Ensure matrix dimensions match.

- Overload scalar multiplication and division operators:
  - Support expressions like `matrix * frac`, `frac * matrix`, and `matrix / frac`.

- Overload the multiplication operator `*` to perform matrix multiplication:
  - Multiply two `FractionMatrix` objects using standard matrix multiplication rules.
  - Ensure valid dimensions.

- Overload the `%` operator to perform element-wise (Hadamard) multiplication between two matrices of the same size.
  - Ensure valid dimensions.

- Implement a method `transpose()` that:
  - Returns a new `FractionMatrix` object which is the transpose of the current matrix.

- Overload the stream insertion operator `<<` to display the matrix:
  - Print the matrix in a readablerow-column format.

Note: You must reuse the `Fraction` and `FractionVector` classes and the methods/operators of those classes from Tasks 1 and 2 for an operation if possible.

Task 4: Test Cases and Demonstration (10 Marks)

Write a `main()` function to:

- Demonstrate all arithmetic operations on `Fraction`.
- Perform vector addition, scalar multiplication, dot product, and compute vector magnitude.
- Perform matrix addition, scalar multiplication, matrix multiplication, and Hadamard product.

Note

For the classes that require them, implement the following:

- **Copy constructor**
- Overload the **assignment operator**
- **Destructor**

Submission Guidelines

- Create a folder named by your ID. Copy your .cpp files in the folder, zip the folder and submit the zip file.

Assessment Criteria

Task	Criteria	Marks
Fraction Class	Correct implementation and operator overloading	20
FractionVector Class	Correct functionality and operator overloading	15
	Proper reuse of Fraction class methods	15
FractionMatrix Class	Correct functionality and operator overloading	20
	Proper reuse of Fraction and FractionVector methods	20
Test Cases and Demonstration	Comprehensive and correct test cases	10