Homework 3: Operator Overloading
Due Date: 10th November, 11:59 PM

Objective
The goal of this assignment is to understand and implement operator overloading for a custom C++ class. You will create a class that represents a **mathematical fraction** (e.g., 1/2) and then overload common operators to make your class intuitive and easy to use.

Background
In C++, you can change how operators like +, -, *, ==, and << behave with your custom objects. This is called operator overloading. The purpose is to make code more readable. For example, instead of writing Fraction f3 = f1.add(f2), you can write the much more natural Fraction f3 = f1 + f2.

Task: The Fraction Class
You will create a **Fraction** class that has a numerator and a denominator. You will then overload arithmetic, comparison, and stream operators to work with Fraction objects.

Part 1: Fraction.h (Class Definition)
Create a header file named **Fraction.h.** Your Fraction class should have the following:

Private Members:
- int numerator;
- int denominator;
- A private helper function: void simplify();
  - This function should be called by constructors and any arithmetic function to ensure fractions are always stored in their simplest form (e.g., 2/4 should be stored as 1/2).
  - It should also handle signs (e.g., 1/-2 should be stored as -1/2, and -1/-2 as 1/2).
- A private helper function: int gcd(int a, int b);
  - This function should calculate the Greatest Common Divisor (GCD) of two numbers. You will need this for your simplify() function. The Euclidean algorithm is a good way to implement this.

Public Members:
- Constructors:
  - Fraction(); (Default constructor: initializes the fraction to 0/1)
  - Fraction(int n); (One-parameter constructor: initializes to n/1)
  - Fraction(int n, int d); (Two-parameter constructor: initializes to n/d)
    - Important: This constructor must handle the case where d is 0. If d is 0, print an error message and initialize the fraction to 0/1 to prevent a divide-by-zero error.
    - This constructor should call simplify() before finishing.

- Overloaded Arithmetic Operators (as member functions):
    - Fraction operator+(const Fraction& other) const;
    - Fraction operator-(const Fraction& other) const;
    - Fraction operator*(const Fraction& other) const;
    - Fraction operator/(const Fraction& other) const;
- Overloaded Comparison Operators (as member functions):
    - bool operator==(const Fraction& other) const;
    - bool operator!=(const Fraction& other) const;
    - bool operator<(const Fraction& other) const;
    - bool operator>(const Fraction& other) const;
- Overloaded Stream Operators (as friend functions):
    - friend std::ostream& operator<<(std::ostream& os, const Fraction& f);
    - friend std::istream& operator>>(std::istream& is, Fraction& f);

Part 2: Fraction.cpp (Class Implementation)

Create a source file named **Fraction.cpp** to implement all the functions and constructors defined in **Fraction.h.**

Part 3: main.cpp (Test Program)

Create a main.cpp file to demonstrate that your class and all its overloaded operators work correctly. Your main should:

1. Include iostream and Fraction.h.
2. Create several Fraction objects using all your constructors.
3. Test all four arithmetic operators. Print the operation and the result.
4. Test all four comparison operators using if statements.
5. Test the stream input operator.
6. Test your edge cases (zero denominator, simplification).

Bonus Challenge (Optional: +3 points)

Implement the following compound assignment and increment operators as member functions:

- Fraction& operator+=(const Fraction& other);
- Fraction& operator++(); (pre-increment, e.g., ++f1)
- Fraction operator++(int); (post-increment, e.g., f1++)

Submission Guidelines

1. Add the following header comment block to the top of `Fraction.h`, `Fraction.cpp`, and **`main.cpp`:**

```
// Name: Your Full Name
// Student ID: YourID
// Assignment: Homework 3 - Operator Overloading
```

2. Create a `.zip` archive containing the following files:

    * `Fraction.h`
    * `Fraction.cpp`
    * `main.cpp`
    * A **screenshot** of your complete program run from `main.cpp`. The screenshot must clearly show:
        * The output from all tests (constructors, arithmetic, comparisons, simplification).
        * The error message from the zero-denominator test.

3. Name your archive file in the format: **Lastname_hw3.zip** (e.g., `Smith_hw3.zip`).
4. Submit the `.zip` file to Canvas