CS-1004 Object Oriented Programming Fall - 2023

[Assignment – 03] **[Total Marks 500]**

Instructions.

- **Plagiarism is strictly prohibited. If any instance of plagiarism is detected, a consequence of receiving a score of zero will be applied to all assignments, quizzes, and homework. Additionally, the matter will be forwarded to the Disciplinary Committee. No exceptions or excuses will be entertained.**
- Combine all your work in one folder. The folder must contain only the **CPP files and header files**.
- Rename the folder as ROLL-NUM_SECTION_NAME (e.g. 22i-1234_A_Ali) and compress the folder as a zip file. (e.g. 22i-1234_A_Ali.zip).
- Do not submit .rar file.
- Submit the .zip file on Google Classroom within the deadline.
- Submission other than Google classroom (e.g. Email etc.) Will not be accepted.
- The student is solely responsible to check the final zip files for issues like corrupt file, virus in the file, mistakenly exe sent.
- If the instructor cannot download the file from Google classroom due to any reason it will lead to zero marks in the assignment.
- Deadline of assignment is **4th November 2023 11:59 PM.**
- Deadline to **submit** assignment is **5th November 2023 11:59 PM. (a whole one day for submission only).**
- Assignments submitted after the deadline will be marked DIRECT ZERO.
- You are supposed to submit your assignment on GOOGLE CLASSROOM (CLASSROOM TAB only, not lab).
- Correct and timely submission of the assignment is the responsibility of every student; hence no relaxation will be given to anyone.
- For timely completion of the assignment, start as early as possible.
- Comments: Comment on your code properly. Write your name and roll number (as a block comment) at the beginning of the solution to each problem.
- You must do proper allocation and deallocation of the memory where necessary.
- All programs must be generic.
- For timely completion of the assignment, start as early as possible.
- Your code should be modular.
- Note: Follow the given instructions to the letter, failing to do so will result in a zero.

Operator Overloading for Rational Numbers in C++

The objective of this assignment is to implement operator overloading for rational numbers in C++. You will create a class called Rational that represents fractions and implement the necessary operators to perform arithmetic operations, comparison operations, and other relevant operations defined below on these fractions.

Instructions:

1. Create a **C++ class named Rational** with the following necessary items.
   - **[a]** Two integer data members: numerator and denominator to represent the fraction.
   - **[b]** Use constructors (all types) and destructors appropriately to manage the object's lifecycle.
   - **[c]** Ensure that the denominator is never zero and handle negative numbers properly. Use function overloading for different constructor options.
   - **[d]** Implement accessors (getters) and mutators (setters) to access and modify private members.
   - **[e]** Use encapsulation to control access to the private members.
   - **[f]** Use member initialization lists for efficient object initialization.
   - **[g]** Utilize const variables and const data members where appropriate.
   - **[h]** Define const member functions to ensure certain functions do not modify the object.
   - **[i]** Demonstrate the use of const objects where applicable.
   - **[j]** Implement static data members and static functions to demonstrate their use within the Rational class.
2. Create a **menu-driven C++ program** that allows users to interact with the Rational class. The menu should offer options for creating Rational objects, and it should provide access to perform all available operators and display results. Ensure that users can choose and perform all available operations using the menu only.
3. Ensure that the program has a clear **3 file structure**, including separate header and source files for the Rational class.
4. **Overload** the following **operators** of the Rational class:
   - **[a]** **Arithmetic Operators:** + (Addition), - (Subtraction), * (Multiplication), and / (Division) to perform basic arithmetic operations.
   - **[b]** **Comparison Operators:** == (Equality), != (Inequality), > (Greater Than), < (Less Than), >= (Greater Than or Equal To), and <= (Less Than or Equal To) to compare Rational objects.
   - **[c]** **Unary Operators:** ++ (Prefix Increment), and -- (Prefix Decrement), - (Unary Minus), + (unary plus), and ! (logical NOT) to perform unary operations.
   - **[d]** **Assignment Operator:** Assignment operator (=) to assign one Rational object to another.
   - **[e]** **Arithmetic Assignment Operators:** += (Addition Assignment), -= (Subtraction Assignment), *= (Multiplication Assignment), and /= (Division Assignment) for arithmetic assignment.
   - **[f]** **Stream Operators:** << (Output Operator) and >> (Input Operator) to display and input Rational objects.
   - **[g]** Implement a member function named **simplify** that simplifies the Rational object by finding the greatest common divisor (GCD) of the numerator and denominator and dividing both by the GCD.
5. In the program, include **test cases to verify the correctness** of your operator overloads and the simplify function.
6. Proper **comment on** your code and **provide explanations** for the major parts of the program.
7. A **brief report** explaining how you implemented the operator overloads and demonstrating the correctness of your code with test cases.

Submission: Submit the following:

1. Your **C++ source code** files.
2. A **brief report** explaining how you implemented the operator overloads and demonstrating the correctness of your code with test cases.

Grading: Your assignment will be graded on the following criteria:

1. Correct implementation of the Rational class with proper operator overloads.
2. Proper handling of constructors, destructors, and object lifecycle management.
3. Effective use of function overloading and constructor overloading.
4. Proper use of encapsulation and access specifiers.
5. Implementation of accessors and mutators.
6. Appropriate use of const variables, data members, and member functions.
7. Demonstrated use of const objects and static members.
8. Correct and tested program functionality, including a menu-driven interface with all available operators.
9. Proper file structure.
10. Code readability, organization, and commenting.