

CSCI205 Data Structures and Algorithms

Lab Assignment

This assignment is a continuation of the Fraction class presented in Chapter 1 of Runestone. It also serves as a gentle introduction to C++ as you have worked with this concept of Fractions and classes before so the idea and logic should be familiar. The assignment attempts to illustrate some of the important differences between Java and C++. Especially the following

- Class structures: header and implementation files
- Overloading of C++ operators
- Passing objects by reference vs by value

All code should compile and run error free. As we progress through the class these requirements will become more stringent as we study

- Warnings
- Dynamic memory allocation and de-allocation

AI Policy: Review the AI policy contained in the course outline. Citations and documentation are **required**. Your submission of this exercise is a testimony to your authorship of all code and comments. You are also stating that you 100% understand all ideas contained herein.

Tasks:

1. Separate the main function and class interface from the class implementation by creating the following files

- a. **Fraction.h** *(class interface – header file)*
- b. **Fraction.cpp** *(class implementation file)*
- c. **main.cpp** *(main function and application logic)*

2. Documentation requirements

- a. Header comments should describe everything a programmer needs to know to **use the function**
- b. Header file documentation should use **Doxygen** style comments. You don't have to install the doxygen tool, just use the formatting rules. This is similar to Javadoc:

https://darkognu.eu/programming/tutorials/doxygen_tutorial_cpp/#documenting-the-code

- c. Implementation file comments describe **how** the function accomplishes its task.

3. Review the document ***Class_with_separate_source_files.pdf*** for an example and brief explanation. Also check the ***learncpp*** article on this organizational technique. This is a different structural approach from Java:

- a. <https://www.learncpp.com/cpp-tutorial/class-code-and-header-files/>

4. Implement the methods ***getNumerator*** and ***getDenominator***

5. Modify the constructor for the Fraction class so that GCD is used to reduce a fraction immediately. Notice that this means the + function no longer needs to reduce. Make the necessary modifications.

6. Overload the following operators: {-, *, /, >, >=, <, <=, !=}

- a. <https://www.learncpp.com/cpp-tutorial/introduction-to-operator-overloading/>

7. Overload ***cout*** such that a Fraction object can be printed in this fashion

- a. Fraction fracObject(1, 2);
- b. cout << fracObject;
- c. Output: 1/2
- d. Mixed fractions should be resolved
 - i. Fraction mixedFraction(10, 3);
 - ii. cout << mixedFraction;
 - iii. Output 3 1/3

- e. <https://www.learncpp.com/cpp-tutorial/overloading-the-io-operators/>

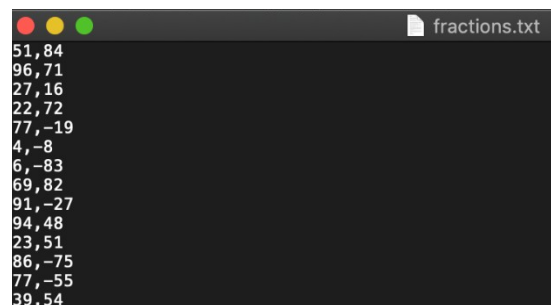
8. Modify the constructor to allow the user to pass a negative denominator so that all of the operators continue to work properly.

9. Overload += keeping in mind that denominators can be negative.

In a separate source code file (**main.cpp** the application) define the **main function** to accomplish the following

1. Define a vector of Fraction objects
2. Open and read the contents of **fractions.txt**. This file is organized into rows containing numerators and denominators separated by commas.

<https://www.tutorialspoint.com/read-data-from-a-text-file-using-cplusplus>



A screenshot of a terminal window titled 'fractions.txt'. The window displays a list of 15 lines, each containing two integers separated by a comma, representing fractions. The fractions are: 51,84; 96,71; 27,16; 22,72; 77,-19; 4,-8; 6,-83; 69,82; 91,-27; 94,48; 23,51; 86,-75; 77,-55; 39,54.

```
51,84
96,71
27,16
22,72
77,-19
4,-8
6,-83
69,82
91,-27
94,48
23,51
86,-75
77,-55
39,54
```

3. For each row in the file construct a Fraction object and add it to the vector. You will need to split the string into the numerator and the denominator. **Research splitting strings.** Cite your sources and include comments about the approach you chose. **Do not include any code that you cannot verbally explain without using references.**
4. Write the function **Fraction find_largest(const vector<Fraction>&)** this function will return the largest fraction. Use your overloaded comparison operators for this task. The signature of this function can be interpreted as
 - a. **vector<Fraction>&:** The use of the address of operator (&) here means that the vector will be passed **by reference** (The address of the vector will be passed and no copy will be made). It's important to remember that you have to be specific with the passing method in C++. Java and Python always passed references to objects as arguments.
 - b. **const:** Including const here means that the original vector cannot be modified (the vector the reference points to when inside this function).
 - c. This function accepts a const reference to a vector of Fraction objects.
5. Write the function **void print(const vector<Fraction>& fractions)** this function will print the fractions to the screen 5 fractions per line.
6. Demonstrate that your overloaded operators function properly by choosing Fractions from the vector and demonstrating **{+, -, /, *}**
7. **main.cpp** should clearly demonstrate the all functionality described above.
8. Push all files to your remote repository before the due date. Use the terminal for these tasks.

RUBRIC

Task	Points
Code compiles and executes with no errors	2
Code is correct, all features implemented and largest fraction is found	20
Class is separated into header and implementation files	5
Code is meaningfully commented in both implementation and interface files	3