**Rational Calculator**

**Part One - Usage & Sample Execution**

The program runs and interacts with the user via the output. The program starts with two pre-initialized Rationals: 1/2 and -1/3.

Upon execution, the program will display the main menu:

1. Enter First Rational  
 2. Enter Second Rational  
 3. Arithmetic Operations  
 4. Equality Testing  
 0. Quit

1. Asks for the numerator then denominator of your first Rational.
2. Asks for the numerator then denominator of your second Rational.
3. Contains a sub-menu of all arithmetic operations. The operations occur in the format:  
    FIRST RATIONAL OPERATION SECOND RATIONAL
4. All equality testing operations. The operations occur in the same format as with arithmetic.

Selecting '0' at the main menu terminates the program.

For the arithmetic and equality operations, simply enter the number corresponding to the operation you wish to perform.

Sample execution (user input is **bolded**):

1. Enter First Rational

2. Enter Second Rational

3. Arithmetic Operations

4. Equality Testing

0. Quit

**1**

First Rational:

Please Enter Numerator - **3**

Please Enter Denominator - **4**

1. Enter First Rational

2. Enter Second Rational

3. Arithmetic Operations

4. Equality Testing

0. Quit

**2**

Second Rational:

Please Enter Numerator - **7**

Please Enter Denominator - **-3**

1. Enter First Rational

2. Enter Second Rational

3. Arithmetic Operations

4. Equality Testing

0. Quit

**3**

Please select an operation:

1. (+)Add Rationals

2. (-)Subtract Rationals

3. (\*)Multiply Rationals

4. (/)Divide Rationals

**3**

3/4 \* -7/3 = -7/4

0.75 \* -2.33 = -1.75

1. Enter First Rational

2. Enter Second Rational

3. Arithmetic Operations

4. Equality Testing

0. Quit

**4**

Please select an equality:

1. (==)Equals

2. (!=)Does no Equal

3. (<)Less Than

4. (<=)Less Than or Equal To

5. (>)Greater Than

6. (>=)Greater Than or Equal To

**4**

3/4 <= -7/3 : false

0.75 <= -2.33 : false

1. Enter First Rational

2. Enter Second Rational

3. Arithmetic Operations

4. Equality Testing

0. Quit

**0**

BUILD SUCCESSFUL (total time: 57 seconds)

**Part Two - Design Decisions**

Dealing with 0 in the Denominator

By restricting user input in the menu via the *Rational\_Error()* exception, any situation of dividing by 0 is avoided altogether. If a zero is entered as the denominator for either the first or second rational, the exception is thrown and the rational is never created. By using pre-initialized rationals with non-zero denominators, there exists no case where it may occur uncaught.

Edge Case for Overflow

When a sufficiently large prime number is in the numerator of either rational, the normalization to reduce will not be effective (if the denominator is not that number). In this case, overflow will occur and an incorrect number will be displayed after calculation. In order to prevent this, one would have to restrict user input to approximately the square root of 2147483647 which would be 46340 .

46340.95000105...

The reason for this is that is specific calculations, numerators and denominators are multiplied. To ensure no overflow, we must consider the case where we are receiving the product of two large numbers. , which approaches but does not exceed the max integer size.

The decision to allow larger numbers was made in the design, although commented code exists to place this restriction and make the code overflow - proof (while making several checks for normalization redundant). This would occur using *Calculation\_Error().*

Normalization

In order to avoid normalizing often, checks are done to determine whether the operation conducted on the two integers in question will exceed the integer maximum of 2,147,483,647.

In the operations *Add* and *Subtract*, a check is done to ensure the absolute values of numerator and denominator of both Rationals are less than 1073741823 (approximately half of the max size of integer). If a number is too large, the *Normal* method is invoked on that rational and it is (hopefully) reduced. The drawback to this approach is that there still exist some cases of unnecessary normalization:

* When adding a large positive Rational to a large negative Rational, resulting in a small number.
  + ex: 123456789/1 + -123456789/1 = 0
* When subtracting a large negative Rational from a large negative Rational, resulting in a small number.
  + ex: -123456789/1 - -123456789/1 = 0
* Any case where the addition or subtraction on a Rational with a numerator / denominator number > 1073741823 would not cause the resulting number to be larger than the maximum size of integer.

In the operations *Multiply, Divide, LessThan, LessEqual, GreaterThan* and *GreaterEqual*, multiplication is conducted on the numerator and denominator of the Rationals. In order to ensure the resulting numbers do not exceed maximum integer value, a check is done to ensure the absolute value of numerator and denominator are < 46340, which is approximately the square root of 2147483647. Should they exceed 46340, that rational is normalized and (hopefully) reduced.