Addition +
Subtraction -
Multiplication *
Division /
Equality ==
Not Equal !=
Greater than >
Less than <
Greater than or Equal >=
Less than or Equal <=

Input is taken in using <<
Output is outputted by >>

*****TEST INPUT*****
fsikram_vanand_Testing.txt file: All the tests are grouped together in this one file

```
2/45 1/45 +
1 1 +
-1/13 2/5 +
4/26 2/13 +
2/5 1/-5 -
5/6 2/3 -
7/8 9/72 -
2/7 1/7 /
-3/4 4/-6 /
6/7 1/2 /
2/11 1/11 *
4/7 7/2 *
9/10 3/8 *
3/5 4/7 + 4/8 - 3/9 * 100 *
3/5 4/7 - 4/8 + 3/9 / 100 /
4/8 8/16 * 4/5 / 2/3 + 2/6 -
4/9 7/9 / 2/4 * 1/7 - 5/6 +
2/7 1/8 >
-5/900 2/900 >
8/10 8/10 >
2/23 2/23 <
2/-29 4/29 <
100/1000 80/1000 <
2/6 2/6 <=
-5/7 2/-3 <=
3/8 1/4 <=
9/64 8/32 <=
2/5 1/5 >=
53/69 53/69 >=
-4/-5 -7/-9 >=
11/73 39/79 >=
3/4 2/3 !=
-2/7 2/-7 !=
5/6 5/6 !=
9/90 45/67 !=
3/4 2/3 ==
2/4 2/4 ==
7/800 2/300 ==
8/9 -8/9 ==
```

Each of these files contain the operators that is being tested. For example, addition contains the tests for the addition operator and so on.

fsikram_vanand_Addition.txt file:
Tests all the addition operators with fractions, whole numbers, and negatives. Prime numbers and non prime numbers in denominator is also tested.
fsikram_vanand_Subtraction.txt file:
Tests all the subtraction operators with fractions and negatives. Prime numbers and non prime numbers in denominator is also tested.
fsikram_vanand_Multiplication.txt file:
Tests all the multiplication operators with fractions and negatives. Prime numbers and non prime numbers in denominator is also tested.
fsikram_vanand_Division.txt file:
Tests all the division operators with fractions and negatives. Prime numbers and non prime numbers in denominator is also tested.
fsikram_vanand_Greater.txt file:
Tests fractions that are equal, less than, and greater than. Also tests negatives. Prime numbers and non prime numbers in denominator is also tested.
fsikram_vanand_GreaterOrEqual.txt file:
Tests fractions that are equal, less than, and greater than. Also tests negatives. Prime numbers and non prime numbers in denominator is also tested.
fsikram_vanand_Less.txt file:
Tests fractions that are equal, less than, and greater than. Also tests negatives. Prime numbers and non prime numbers in denominator is also tested.
fsikram_vanand_LessOrEqual.txt file:
Tests fractions that are equal, less than, and greater than. Also tests negatives. Prime numbers and non prime numbers in denominator is also tested.
fsikram_vanand_NotEqual.txt file:
Tests fractions that are equal, less than, and greater than. Also tests negatives. Prime numbers and non prime numbers in denominator is also tested.
fsikram_vanand_Equal.txt file:

Tests fractions that are equal, less than, and greater than. Also tests negatives. Prime numbers and non prime numbers in denominator is also tested.
fsikram_vanand_MultipleOperators.txt file:
Tests if multiple operators work if all inputted on one line. Also tests prime numbers and non prime numbers in denominator.

*****SUMMARY*****
This program implements a class for rational numbers. The usual arithmetic operators, +, -, *, /, >, >=, <, <=, !=, == are overloaded to perform simple math which presents fractions in simplified form and comparisons which displays true or false. The stream insertion and extraction (<< and >>) operators are included to read and print out rational numbers.

******HOW TO RUN IT*****
In order to run it, you first have to compile the program. You can compile it by typing in the word make into the terminal. Then, you need to type in the name of the file and followed by the name of the input files which can amount to any number. The following is an example command written out to run the program: ./PA5 fsikram_vanand_Addition.txt fsikram_vanand_Subtraction.txt fsikram_vanand_Multiplication.txt fsikram_vanand_Division.txt fsikram_vanand_Greater.txt fsikram_vanand_Less.txt fsikram_vanand_GreaterOrEqual.txt fsikram_vanand_LessOrEqual.txt fsikram_vanand_NotEqual.txt fsikram_vanand_Equal.txt fsikram_vanand_MultipleOperators.txt

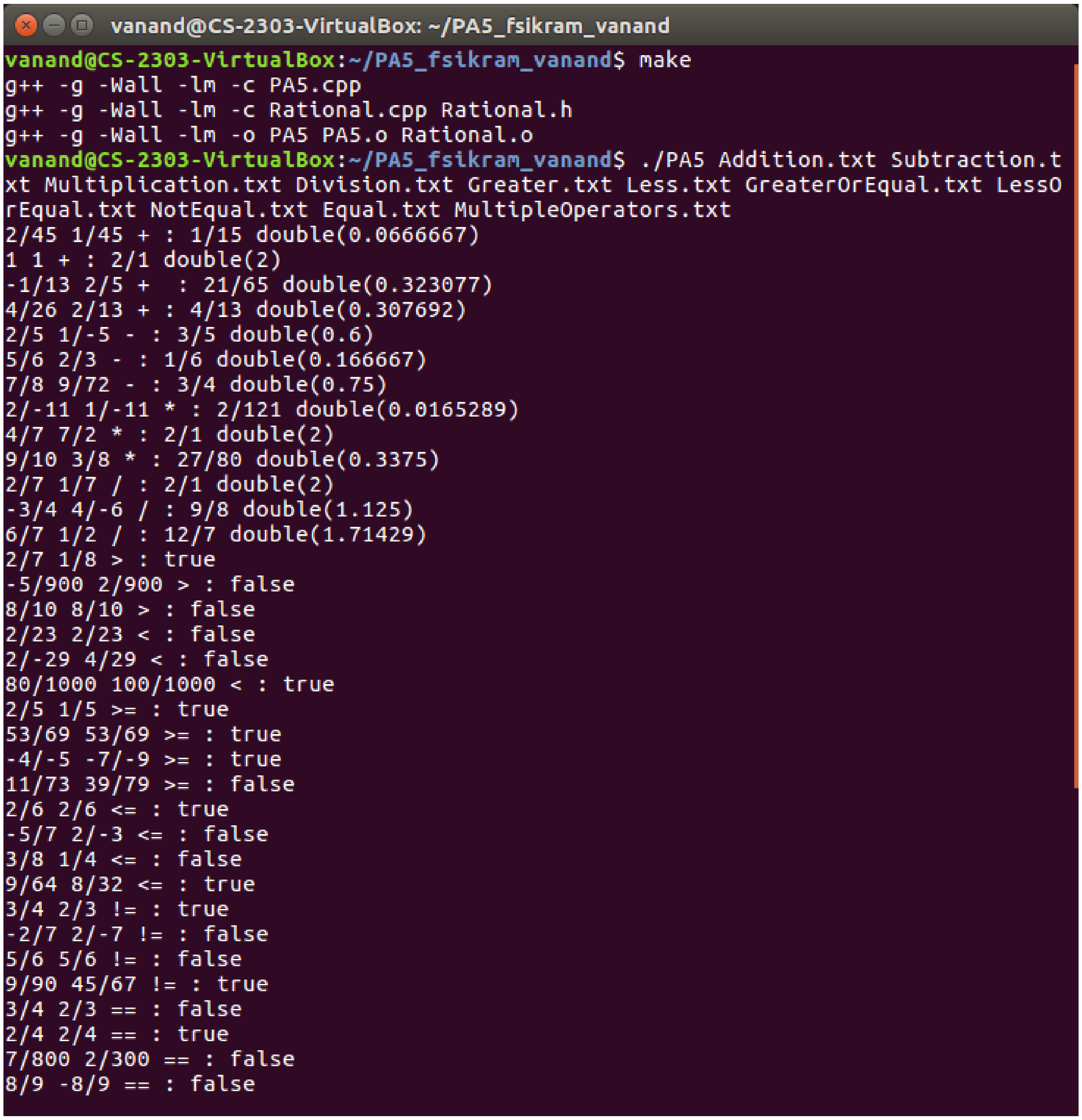
*****PROBLEMS*****
We did encounter some problems in this program. We first had trouble with starting the assignment, but then we were able to read the C++ book and figure out where to start. We also didn’t know how to implement the friend function but we were able to read the book to figure out how it worked and implement it in our function.

*****ALGORITHM*****
The rational class has a few constructors. The first constructor is a default constructor that sets the numerator equal to 0 and denominator equal to 1, which is the default fraction. The results from the operations are then updated from this. The next constructor looks at the case of the denominator being less than 0 and equal to 0, in which case it throws an error since you cannot have 0 in the denominator. The next constructor takes care of whole numbers and puts them over 1. The next constructor simplifies the numerator and denominator. Next are the operators that do its own comparisons. They take the numerator if the first fraction and multiply it with the denominator of the second fraction then uses one of the comparison operators to compare with the product of the second fraction’s numerator with the first fraction’s denominator. The next operations are the math operations. The functions performs the math for each the numerator and denominator for that specific function, puts together the resulting numerator and denominator, and simplifies that resulting fraction. For example, for the addition operator, the numerator is found by multiplying the first fraction’s numerator by the second fraction’s denominator and then adding that to the product of the first fraction’s denominator and the second fraction’s numerator. The denominator is found by multiplying the second and first

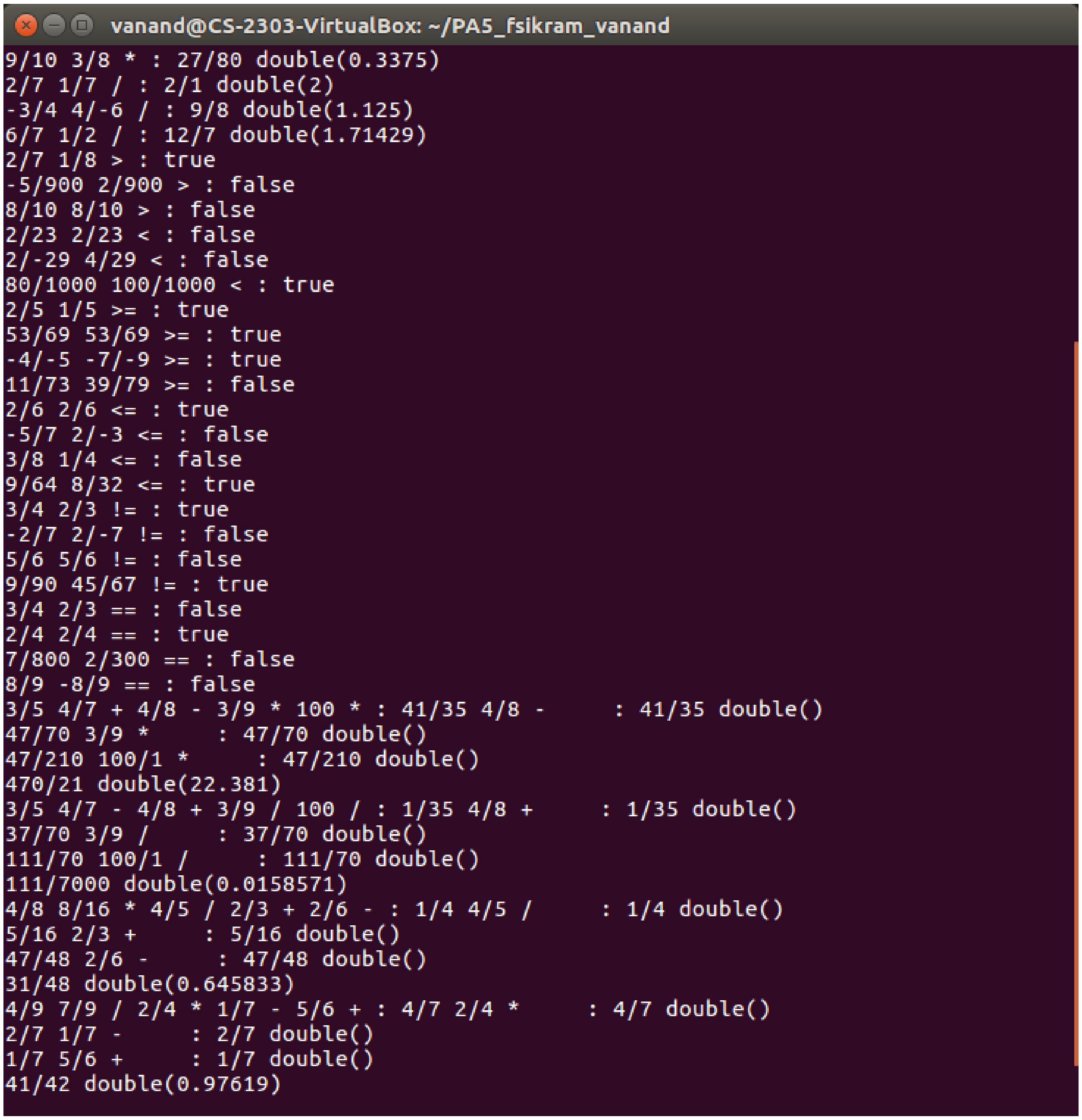
fraction’s denominator. Then, these are put in fraction form and simplified. The simplify function simplifies the numerator and denominator by finding the greatest common denominator through a subtraction method. All of these constructors are defined in the rational.h header file. The test program processes command lines, opens and closes input files, scans, parses, and evaluates input lines.

*****RESULTS FROM TESTING*****
We tested our program but having an input file that has the different kinds of fractions and implemented some arithmetic functions on it to make sure it was doing the math and comparisons correctly. The input file has 3 examples of each type of arithmetic operation. For example, for addition the input file may have: ¾ ½ +

Outputs from testing:
(Please look on next two pages)



```
vanand@CS-2303-VirtualBox:~/PA5_fsikram_vanand$ make
g++ -g -Wall -lm -c PA5.cpp
g++ -g -Wall -lm -c Rational.cpp Rational.h
g++ -g -Wall -lm -o PA5 PA5.o Rational.o
vanand@CS-2303-VirtualBox:~/PA5_fsikram_vanand$ ./PA5 Addition.txt Subtraction.t
xt Multiplication.txt Division.txt Greater.txt Less.txt GreaterOrEqual.txt LessO
rEqual.txt NotEqual.txt Equal.txt MultipleOperators.txt
2/45 1/45 + : 1/15 double(0.0666667)
1 1 + : 2/1 double(2)
-1/13 2/5 +  : 21/65 double(0.323077)
4/26 2/13 + : 4/13 double(0.307692)
2/5 1/-5 - : 3/5 double(0.6)
5/6 2/3 - : 1/6 double(0.166667)
7/8 9/72 - : 3/4 double(0.75)
2/-11 1/-11 * : 2/121 double(0.0165289)
4/7 7/2 * : 2/1 double(2)
9/10 3/8 * : 27/80 double(0.3375)
2/7 1/7 / : 2/1 double(2)
-3/4 4/-6 / : 9/8 double(1.125)
6/7 1/2 / : 12/7 double(1.71429)
2/7 1/8 > : true
-5/900 2/900 > : false
8/10 8/10 > : false
2/23 2/23 < : false
2/-29 4/29 < : false
80/1000 100/1000 < : true
2/5 1/5 >= : true
53/69 53/69 >= : true
-4/-5 -7/-9 >= : true
11/73 39/79 >= : false
2/6 2/6 <= : true
-5/7 2/-3 <= : false
3/8 1/4 <= : false
9/64 8/32 <= : true
3/4 2/3 != : true
-2/7 2/-7 != : false
5/6 5/6 != : false
9/90 45/67 != : true
3/4 2/3 == : false
2/4 2/4 == : true
7/800 2/300 == : false
8/9 -8/9 == : false
```



vanand@CS-2303-VirtualBox: ~/PA5_fsikram_vanand

```
9/10 3/8 * : 27/80 double(0.3375)
2/7 1/7 / : 2/1 double(2)
-3/4 4/-6 / : 9/8 double(1.125)
6/7 1/2 / : 12/7 double(1.71429)
2/7 1/8 > : true
-5/900 2/900 > : false
8/10 8/10 > : false
2/23 2/23 < : false
2/-29 4/29 < : false
80/1000 100/1000 < : true
2/5 1/5 >= : true
53/69 53/69 >= : true
-4/-5 -7/-9 >= : true
11/73 39/79 >= : false
2/6 2/6 <= : true
-5/7 2/-3 <= : false
3/8 1/4 <= : false
9/64 8/32 <= : true
3/4 2/3 != : true
-2/7 2/-7 != : false
5/6 5/6 != : false
9/90 45/67 != : true
3/4 2/3 == : false
2/4 2/4 == : true
7/800 2/300 == : false
8/9 -8/9 == : false
3/5 4/7 + 4/8 - 3/9 * 100 * : 41/35 4/8 -      : 41/35 double()
47/70 3/9 *      : 47/70 double()
47/210 100/1 *      : 47/210 double()
470/21 double(22.381)
3/5 4/7 - 4/8 + 3/9 / 100 / : 1/35 4/8 +      : 1/35 double()
37/70 3/9 /      : 37/70 double()
111/70 100/1 /      : 111/70 double()
111/7000 double(0.0158571)
4/8 8/16 * 4/5 / 2/3 + 2/6 - : 1/4 4/5 /      : 1/4 double()
5/16 2/3 +      : 5/16 double()
47/48 2/6 -      : 47/48 double()
31/48 double(0.645833)
4/9 7/9 / 2/4 * 1/7 - 5/6 + : 4/7 2/4 *      : 4/7 double()
2/7 1/7 -      : 2/7 double()
1/7 5/6 +      : 1/7 double()
41/42 double(0.97619)
```