The goal of this lab is to give you more practice in using functions that return values through the return statement and through the parameter-list. You will complete a program that implements a simple fraction calculator. When the program is completed it should run according to the example below.

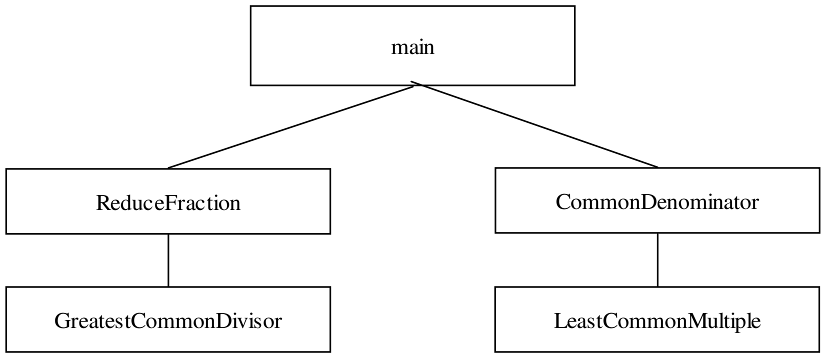
```
Add, subtract, multiply & divide – positive fractions only
Enter ‘0/0 + 0/0’ to quit.

>1/2 + 3/4
2/4 + 3/4 = 5/4

>2/16 + 29/32
4/32 + 29/32 = 33/32

> 1/7 + 1/5
5/35 + 7/35 = 12/35

>1/2 – 1/3
3/6 – 2/6 = 1/6

>200/100 * 25/50
2/1 * 1/2 = 1/1

>1/2 * 3/4
1/2 * 3/4 = 3/8

>1/2 / 3/4
1/2 / 3/4 = 2/3

>0/0 + 0/0
```

Note that the program will add, subtract, multiply, and divide positive fractions only. For subtraction, the lager fraction must be given first so that the answer will be positive also. All answers will be fractions in lowest terms. For purposes of simplifying this assignment, a whole number such as 5 will be written as 5/1.

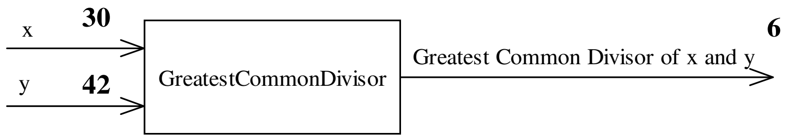
On the second page is a structure chart that show how the functions that you will write will be called by one another. Also given are some diagrams that illustrate what each of the four missing functions is to do. On the third page is a brief description for each function that restates in English what the diagrams are saying. Pages 4 & 5 contain flowcharts for the GCD (Greatest Common Divisor) and LCM (Least Common Multiple) algorithms. Page 6 contains the code for the main function of fraction.cpp that is available at \\cs1\Classes\Comp170\fraction.cpp. Copy this file to your computer so that you can complete the program.

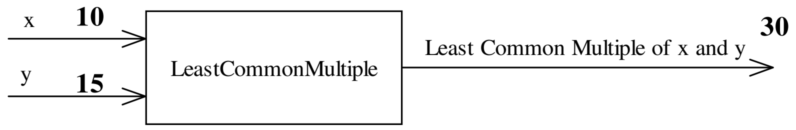
This lab will be due at the beginning of the next class period. Complete the four functions and submit your working program to Easel (http://cs.harding.edu/easel/). Make sure your name and date are documented at the top of the program.

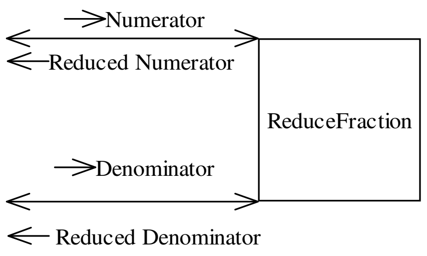
Structure Chart

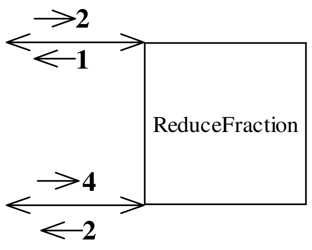


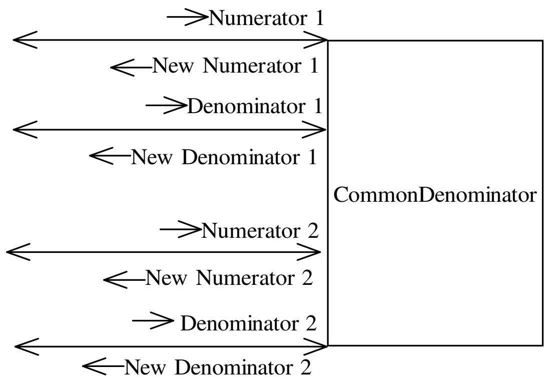
Function Diagrams

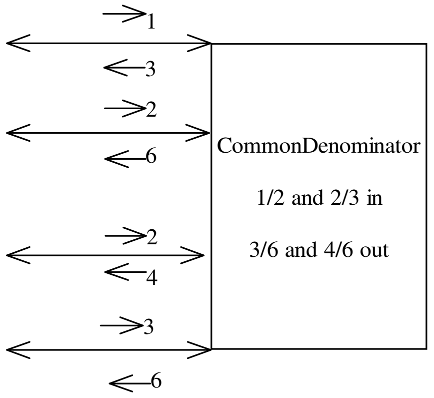












Function Descriptions:

The **GratestCommonDivisor** function receives two integers from the calling program and returns a single integer that is their GCD. For example

```
x = GreatestCommonDivisor(30, 42);
```

will result in a value of 6 for `x`. It is assumed that all integers are positive and are within the normal range for the int data type.

The **LeastCommonMultiple** function receives two integers form the calling program and returns a single integer that is their least common multiple. For example

```
x = LeastCommonMultiple(10, 15);
```

will result in a value of 30 for `x`. It is assumed that all integers are positive and are within the normal range for the int data type.

The **ReduceFraction** function receives two integers from the calling program, and it is assumed that the first integer is the numerator and that the second integer is the denominator of the fraction. The ReduceFraction function calls the GreatestCommonDivisor function and uses the information returned to alter both the numerator and denominator so that the values sent back to the calling program are an equivalent fraction to the one received, but in lowest terms. For example

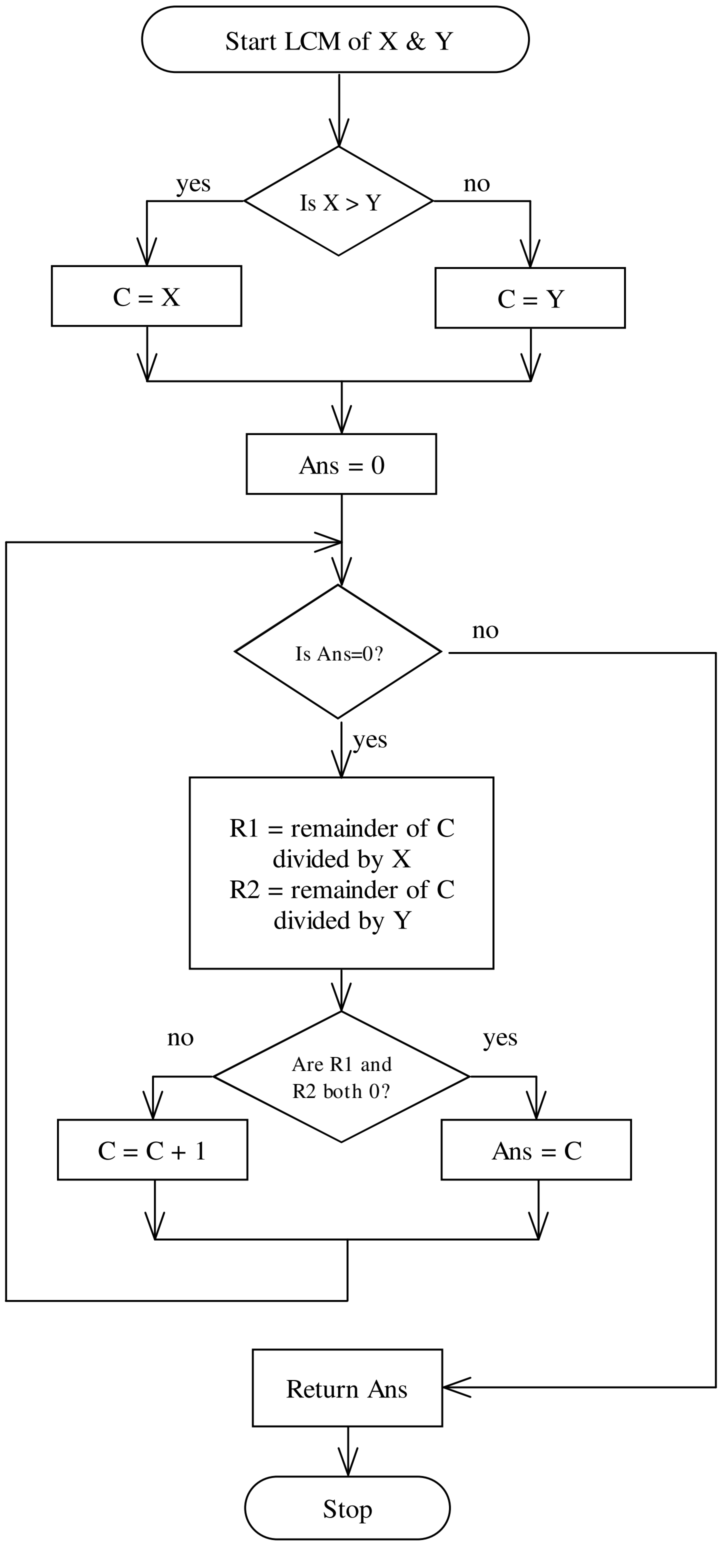
```
x = 2;
y = 4;
ReduceFraction(x, y);
```

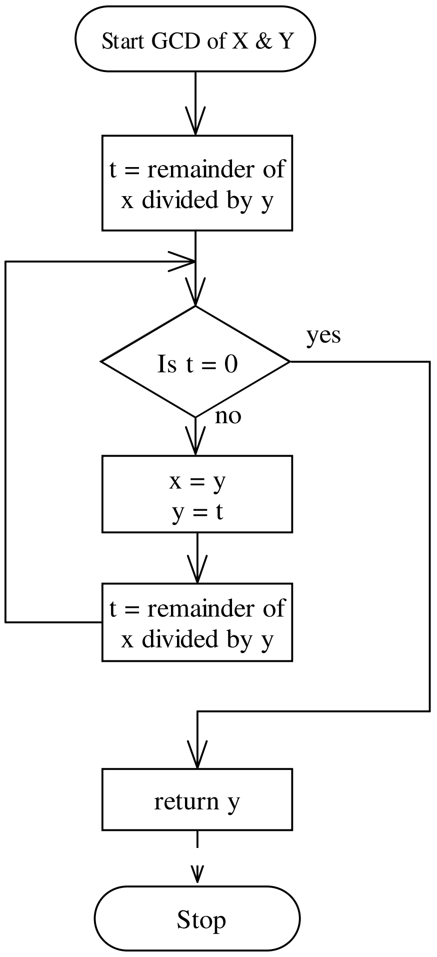
will result in `x` being changed to 1 and `y` being changed to 2 since the original data represents 2/4 and the lowest terms version of 2/4 is 1/2.

The **CommonDenominator** function receives four integers from the calling program, and it is assumed that the first two integers represent the numerator and denominator of one fraction and that the second two integers represent the numerator and denominator of a second fraction. This function will call the LeastCommonMultiple function to find a common denominator and then alter both fractions so that the values returned to the calling program are equivalent fractions with a common denominator. For example

```
numerator1 = 1;
numerator2 = 2;

denominator1 = 2;
denominator2 = 3;

CommonDenominator(numerator1, denominator1, numerator2, denominator2);
```

will result in `numerator1` being changed to 3 and `denominator1` being changed to 6. It will also result in `numerator2` being changed to 4 and `denominator2` being changed to 6. This is due to the fact that the original data represents 1/2 and 2/3 and the commonly denominated equivalents are 3/6 and 4/6.


Start LCM of X & Y
Is X > Y
yes
no
C = X
C = Y
Ans = 0
Is Ans=0?
no
yes
R1 = remainder of C divided by X
R2 = remainder of C divided by Y
Are R1 and R2 both 0?
no
yes
C = C + 1
Ans = C
Return Ans
Stop


Start GCD of X & Y
t = remainder of x divided by y
Is t = 0
yes
no
x = y
y = t
t = remainder of x divided by y
return y
Stop

```
//your name and date here!

#include <iostream>
using namespace std;

//  Missing functions go here!

void main()
{
       int num1, den1, num2, den2, newnum, newden;
       char slash1, slash2, op;

       cout << "\nFraction Calculator\n\n";
       cout << "Add, subtract, multiply & divide - positive fractions only\n";
       cout << "Enter '0/0 + 0/0' to quit.\n";

       // Input will be assumed to be in correct form for simplification
       // Input data before loop in case they want to exit right away

       cout << "\n> ";
       cin >> num1 >> slash1 >> den1 >> op >> num2 >> slash2 >> den2;
       while (num1 + den1 + num2 + den2 > 0)
       {
              // Reduce both fractions to keep integers as small as possible
              ReduceFraction(num1, den1);
              ReduceFraction(num2, den2);
              switch (op)
              {
                     case '+':
                        // Find common denominator and add
                        CommonDenominator(num1,den1,num2,den2);
                        newnum = num1 + num2;
                        newden = den1;
                        break;
                     case '-':
                        // Find common denominator and subtract
                        CommonDenominator(num1,den1,num2,den2);
                        newnum = num1 - num2;
                        newden = den1;
                        break;
                     case '*':
                        // Multiply numerators and multiply denominators
                        newnum = num1 * num2;
                        newden = den1 * den2;
                        break;
                     case '/':
                        // Invert and multiply
                        newnum = num1 * den2;
                        newden = den1 * num2;
              }
              // Reduce the answer to lowest terms
              ReduceFraction(newnum, newden);

              // Output the results
              cout << num1 << "/" << den1 << " " << op << " ";
              cout << num2 << "/" << den2 << " = ";
              cout << newnum << "/" << newden << endl;

              // Input data for next iteration of loop
              cout << "\n> ";
              cin >> num1 >> slash1 >> den1 >> op >> num2 >> slash2 >> den2;
       }
}
```