MSIM 508

Introduction to Game Development

Programming Assignment #1

January 25, 2017

Mark Brosche

Major: MSVE - Master of Engineering

Online Distance Graduate - Outside Virginia (Saratoga Springs, NY)

[mbros006@odu.edu](mailto:mbros006@odu.edu)

407-325-2818

# **INTRODUCTION**

One way of finding a given integer’s prime factors involves dividing said number by the lowest number that divides cleanly into it (i.e. remainder = 0) until it doesn’t anymore and moving on to the next lowest number, until what’s left of the given number settles out to 1. For smaller numbers, the task is easier.

To find the greatest common factor (GCF) of two numbers, one of the simplest methods is to multiply together all of the factors they have in common. Depending on the numbers chosen and their relationship to each other this can also be easier or harder than just doing math in your head.

The least common denominator (LCD, or least common multiple) is the smallest number that each given number divides cleanly into. To arrive at the LCD using the GCF, one can divide the multiple of the two numbers by their greatest common factor.

My goal for this assignment is to practice using and experimenting with some of the basic features of C# to implement a script that calculates the GCF and LCD and interfaces though the console.

# **PROGRAM DESIGN**

Files:

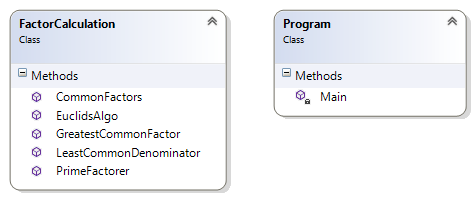
Program.cs

# **CONTROLS**

The program interface uses the console window and keyboard. Instructions are given in the console window.

# **CODE ARCHITECTURE**

**Class Diagram:**



**Key Attributes (variables) and methods:**

**Main():**

Variables:

aFactors - List<int> for the first given number’s prime factors.

bFactors - List<int> for the second given number’s prime factors.

cFactors - List<int> for the common prime factors of the two given numbers.

GCF – int that holds the returned value of the GreatestCommonFactor method.

**FactorCalculation Class:**

Variables:

primeFactors - List<int> used in the PrimeFactorer method, the elements of which are returned.

comFactors - List<int> used in the CommonFactors method, it is used to store the common factors of the lists passed into the function and is returned.

Methods:

PrimeFactorer – a static List<int> function that takes an int argument and returns a List<int> of the prime factors.

CommonFactors – a static List<int> function that takes two List<int> arguments and returns one List<int> containing the common terms shared by the input arguments.

GreatestCommonFactor – a static int function that takes a List<int>, and two ints and returns an int. The List<int> is of primary importance for the calculation, the other arguments are used to print a message to the console.

LeastCommonDenominator – a static void function that takes three ints for arguments and returns none.

EuclidsAlgo – a static int function that takes two ints as arguments and returns an int.

# ASSETS

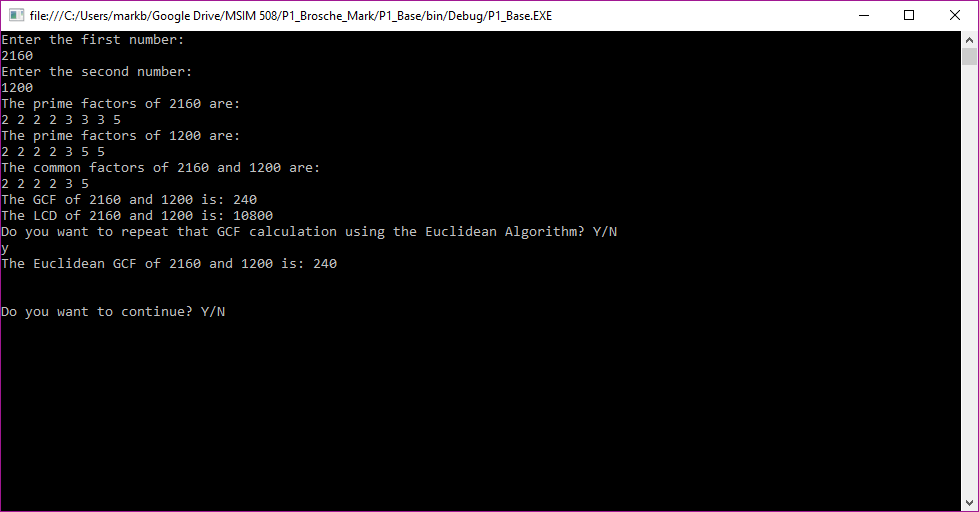
There were no assets used in this programming assignment.

# RESULTS

1. The first task was to read and understand the base code and C# I/O operations
   1. ANSWER: Yes, it was immediately clear, the usefulness of the parsing method to find ints within strings. I also reused the code to ask a user to continue by pressing y or Y to test the Euclidean Algorithm, and any other k to exit.
2. The second task was to declare two List<int> objects to store prime factors of integers, a and b.
   1. ANSWER: See lines 131 and 137 in Program.cs
3. The third task was to write an algorithm to implement the factorization process. The base program defined an integer array variable named primes to represent the prime numbers less than 100. You can use this variable directly in your factorization. However, this variable is not absolutely needed. If this variable (the information about prime numbers less than 100) is not available, how do you implement the factorization process? 3 Extra points for implementations without using this variable.
   1. ANSWER: To factor a given number with the array of primes, simply step through the array with a while loop nested in a for loop and populating a List<int> with the array value each time the modulus of the given number by an array value is 0.
   2. ANSWER: I chose to implement factorization without the array of primes which required changing the for-loop condition to step through as many numbers as necessary until the given number ‘num’ is equal to 1. This removes the cap of 100 from the available list of numbers for a user to choose from (which I deleted from the base conditions).
4. The fourth task is to write an algorithm to find the GCF of two numbers based on their prime factors. For MSIM 508, also define the Euclidean Algorithm (3 Extra points for implementing)
   1. ANSWER: My algorithm relies on two methods, one to assemble a 3rd list with the common factors and the second to multiply each element of the 3rd list together using a for-loop.
   2. ANSWER: The Euclidean algorithm is considered the most efficient method of finding the GCF of two numbers, and is implemented practically by use of the modulus operator. Over as many iterations as necessary, take the modulus of the larger number by the smaller number until it can reduce no further, and the GCF is the result.
5. The fifth task is to write an algorithm to find the LCD of two numbers based on their GCF and prime factors.
   1. ANSWER: I chose to implement the method of determining the LCD by multiplying the given integers a and b and dividing them by their GCF.
6. The sixth task is to write the code in such a way that it prints the results of tasks to the console as shown in the example.
   1. ANSWER: Please execute the code.
7. The seventh task is to discuss other implementations or improvements.
   1. Provided in the Conclusion.

# IMPORTANT SCREEN CAPTURES

The is also a short video in the folder titled: P1\_Brosche\_Mark\_Video.mp4



# CONCLUSION AND DISCUSSION

This assignment is probably my first real programming effort in many years (and I’ve only danced around it even then) so it’s safe to say I’ve learned quite a bit. Getting the hang of creating a separate class for methods and learning to call them was a great exercise in modularity and creating a semi-library. As for the math, that part was pretty easy. I had difficulty understanding how to use the different methods available to a list<T> and got hung up mostly on trying to found out whether two lists had common elements and preventing them from being read multiple times.

I would say the code could be alternatively implemented with other methods. For example, the LCD can be calculated by multiplying the highest power of each common factor together. This in in turn would enable the calculation of more than two numbers. Additionally, recursive code for computing the Euclidean algorithm could be used. My understanding of the use of recursive code requires a bit more experience to implement gracefully however.

I spent roughly 9 hours on this assignment.