COP 3223 Program #7: Number Theory!

Your boss at Pizza Shack has given you a break for the week. Instead of writing programs for him, you get to write programs about your secret love: NUMBER THEORY!!! You remind all of your friends that Gauss once said, "Mathematics is the queen of sciences and number theory is the queen of mathematics."

Program A: Prime Factorization (primefact2.c)

You are excited that Arup showed you a prime factorization program in class, but you think he hasn't done a very good job. For example, if you ask his program to prime factorize 96, it spits out the following output:

```
The prime factorization of 96 is 2 x 2 x 2 x 2 x 2 x 3.
```

You think that it's silly that his program repeats 2 four times. Instead, you think that his program should list the prime factorization as you've seen it in mathematics class, with exponents. For this example, you prefer the output to be:

```
The prime factorization of 96 is 2^5 x 3^1.
```

You only want to list the prime numbers that divide into the original at least once, and for each one, you want to list the number of times it divides in as the corresponding exponent.

Edit the program primefact.c shown in the lectures so that it outputs in this manner.

Input Specification

1. The single input value will be a positive integer in between 2 and 10,000,000.

Output Specification

Output the prime factorization of the number specified, listing each unique prime factor and the corresponding exponent in the format described able.

Sample Run (User input in bold and italics)

```
Please enter a number.
96
The prime factorization of 96 is 2^5 x 3^1.
```

Sample Run (User input in bold and italics)

```
Please enter a number.
169
The prime factorization of 169 is 13^2.
```

Sample Run (User input in bold and italics)

```
Please enter a number.
630
The prime factorization of 630 is 2^1 x 3^2 x 5^1 x 7^1.
```

Program B: Euclid's Algorithm (gcd.c)

In any number theory class, one of the first algorithms discussed is Euclid's Algorithm to find the greatest common divisor of two integers. A slow way to solve the problem is simply try dividing all possible numbers into the two given numbers (using mod) and find the largest divisor that leaves no remainder when dividing into both. For example, the greatest common divisor of 12 and 18 is 6, since 6 divides evenly into both and no larger integer does. Euclid's Algorithm works as follows:

1. Divide the first number into the second number, keeping the remainder.
2. If this remainder is 0, great, the second number is the desired gcd. You are done!
3. Otherwise, reassign the second number as the first one and make the remainder the second number and go back to step 1.

Here are two examples worked out:

gcd(351, 198):
351 % 198 = 153
198 % 153 = 45
153 % 45 = 18
45 % 18 = 9
18 % 9 = 0, so our desired gcd is 9.

gcd(142, 42):
142 % 42 = 16
42 % 16 = 10
16 % 10 = 6
10 % 6 = 4
6 % 4 = 2
4 % 2 = 0, so our desired gcd is 2.

Write a program that asks the user to enter two positive integers and prints out their gcd.

Input Specification

1. The integers will be positive integers less than 1,000,000,000.

Output Specification

Please print out the desired gcd using the format shown in the sample below.

Sample Program Run (User Input in Bold and Italics)

```
For which two integers do you want the gcd?
351 198
GCD(351, 198) is 9.
```

Sample Program Run (User Input in Bold and Italics)

```
For which two integers do you want the gcd?
42 142
GCD(42, 142) is 2.
```

Program C: Primitive Pythagorean Triples (triples.c)

In your number theory course, you learned about Pythagorean Triples. These are side lengths of right triangles that are all integers. For example, (3, 4, 5) is a Pythagorean Triple, since there is a right triangle with sides 3, 4 and 5. Also, (12, 16, 20) is a Pythagorean Triple. A primitive Pythagorean Triple is one such that the gcd of any pair of the values in the triple is 1. Any non-primitive Pythagorean Triple can be created by taking a primitive one and multiplying each of the values in the primitive triple with a constant. For example, with the primitive triple (3, 4, 5), if we multiply each value by 4, we create the triple (12, 16, 20).

In this program, you will generate a set of primitive Pythagorean Triples noting that all primitive triples take on a specific format:

For all primitive Pythagorean Triples, (a, b, c), there exist positive integers x and y, with $x > y$, such that:

$a = x^2 - y^2$
$b = 2xy$
$c = x^2 + y^2$

The only catch is that some of the ordered pairs (x, y) that we can select can generate non-primitive triples. For example, when we try $x = 3$, $y = 1$, we generate the triple (6, 8, 10). (Notice that depending on what values of x and y we plug in, it's not always the case that a will be less than b. Sometimes it is, sometimes it isn't.)

It turns out that the restriction on x and y is relatively simple. Both of those values can NOT be odd (one of them must be) AND gcd(x, y) must equal 1.

Knowing these two restrictions, we can write a program that generates all primitive Pythagorean Triples with an upper bound on the value of x used to generate them. Basically, we run a double for loop through each value of x and y, checking to make sure that exactly one is odd and that their gcd is 1. If this check passes, then we just print out the corresponding Pythagorean Triple according to the formula above.

Input Specification

1. The maximum value for x will be a positive integer in between 2 and 100.

Output Specification

Print out all of the triples, one per line. **Within each triple, print the three values in ascending order, separated by commas.**

Sample Program Run (User Input in Bold and Italics)

```
What is the maximum bound on x?
6
3, 4, 5
5, 12, 13
8, 15, 17
7, 24, 25
20, 21, 29
9, 40, 41
12, 35, 37
11, 60, 61
```

Deliverables

Please submit three separate **.c files** for your solutions to these problems via WebCourses by the designated due date:

Program A: **primefact2.c**
Program B: **gcd.c**
Program C: **triples.c**