1 Exercise: The Euclidean Algorithm

Finding the greatest common divisor of two numbers is an interesting problem. There is an algorithm, called the *Euclidean Algorithm*. Assuming we want to find the greatest common divisor of two numbers a and b, the algorithm is as follows:

1. Divide the larger number by the smaller number, and find the remainder.

2. If the remainder is zero, the smaller number is the greatest common divisor of the two original numbers. Stop.

3. Replace the greater of the two numbers with the remainder. Go back to step 1.

This surely looks almost magical. Let us see this at work. Let us use the numbers 108 and 30. Here is how the calculation goes:

Divide 108 by 30. The remainder is 18. It is not zero, so we continue and replace 108 with 18.

Divide 30 by 18. Remainder is 12. Again nonzero, so we replace 30 by 12 and continiue.

Divide 18 by 12. The remainder is 6. Replace 18 by 6 and continue.

Divide 12 by 6. The remainder *is* zero. So the smaller of the two numbers, 6, is the greatest common divisor of 108 and 30 is 6. You can check that this is true by direct factorization of the two numbers.

We will use this algorithm in order to construct a better program than last time to reduce a fraction to its lowest terms. First, you should write a function that takes two `int` arguments, and returns an `int` called `gcd`. Then, call this function from within `main` to reduce a user-entered fraction to its lowest terms quickly.

2 Exercise: A Number Curio

A "number curio" is an interesting relation among numbers which looks purely accidental an interesting. Here is one: There are four digits a, b, c, d such that

$$a^b \cdot c^d = abcd$$

Obviously, the exercise is to write a program that will find this number. There really is such a number, and there is only one such number!

In order to solve this problem, you need to be able to take powers. Here is how: First, you must have

```
#include <math.h>
```

at the top of your code next to the `#include <stdio.h>`. Then, you can use the `pow` function:

```
double pow(double x, double y)
```

The function returns x^y as a `double`.

3 Exercise: Testing for Primality

In this exercise, we will write a function called `is_prime`, which takes one integer argument, and returns 1 (as an integer) if it is, and a 0 if it is not prime.

How do you test (for certain) whether a number is prime? The only certain test known is to check whether the number is divisible by any numbers up its square root. (If it is divisible by a number greater than the square root, the other factor must be less than the square root.)

In order to write this function, you will need to use the square-root function from the math library. The function is as below:

```
double sqrt(double x)
```

4 Exercise: Testing the Goldbach Conjecture

The Golbach conjecture states that any even number greater than 2 can be written as a sum of two prime numbers (they may be the same). For instance:

```
4 = 2 + 2
6 = 3 + 3
8 = 5 + 3
10 = 5 + 5
12 = 7 + 5
14 = 7 + 7
16 = 13 + 3
18 = 13 + 5
20 = 13 + 7
22 = 11 + 11
24 = 11 + 13
```

This is called a "conjecture" because it has not been proven, and most mathematicians think there is not much hope for its proof either.

The exercise is simply stated: Write a program that will test the Goldbach conjecture for even integers up to a million. For this, you should seriously think about using the `is_prime` function you have written in the previous exercise.