1 Exercise: Reduce to Lowest Terms

```
#include <stdio.h>

int main(void)
{
  int numerator=1;
  int denominator=1;
  int limit;
  int i;

  printf("Please enter the numerator: ");
  scanf("%d", &numerator);
  printf("Please enter the denominator: ");
  scanf("%d", &denominator);

  printf("%d/%d", numerator, denominator);

  limit = numerator > denominator ? denominator : numerator;

  for(i = 2; i <= limit; i++) {
    if (numerator % i == 0 && denominator % i == 0) {
      numerator /= i;
      denominator /= i;
    }
  }

  printf(" = %d/%d\n", numerator, denominator);

  return 0;
}
```

The intended use of the above program is to reduce a fraction to its lowest terms. Type it in to your computer and run it. Check that the output is correct for, say (5, 10).

Now, for each of the following numerator-denominator pairs, run the program once more: (3, 6); (30, 42); (324, 567); (256, 512). Note for each pair whether or not the code correctly reduces the fraction to its lowest terms.

Find out why the program fails in some cases, and does not fail in others. Fix the program.

2 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.

Use this algorithm in order to construct an efficient program that reduces 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.

3 Exercise: Up Up and Down

An algorithm (or, a game) is defined by the following rules:

1. Start with a positive integer.
2. If the number is 1, stop.
3. If it is divisible by 2, divide it by 2. Go back to step 2.
4. If not, multiply the number by 3 and add 1. Go back to step 2.

As an example, assume we start with 3. Here is the sequence one obtains:

3 10 5 16 8 4 2 1

In this case, it took 7 steps to end. How many steps it takes to reach 1 depends on the number we start with. For instance, if the "seed" is a power of 2, the number of the steps will be very few (compared to similarly sized numbers).

The following code is written to do just that, count the number of steps, and print the numbers.

```
#include <stdio.h>

int main(void) {
  int number=3;
  int steps=0;

  printf("Please enter the seed: ");
  scanf("%d", &number);

  while (number != 1) {
    printf("%d ", number);
    steps++;

    if (number % 2 == 0) {
      number /= 2;
    } else {
      number = 3*number + 1;
    }
  }
```

```
  printf("1\n");
  printf("Steps: %d\n", steps);

  return 0;
}
```

Type the program in. Run the program for various values of the initial number. Then, modify the program to find which integer less than 1000 takes the most number of steps to reach 1.

4 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`.

5 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)
```

6 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.