CS 142 Function & Loop Problems

1. A right triangle can have sides that are all integers. A set of three integer values for the sides of a right triangle is called a Pythagorean triple. These three sides must satisfy the relationship that the square of the hypotenuse is equal to the sum of the squares of the other two remaining sides of the triangle. Find all Pythagorean triples for side1, side2, and hypotenuse all no larger than 500. Use a triple-nested for loop that tries all possibilities and prints only the ones that are Pythagorean triples. This is an example of **brute-force computation**, a technique where you just try all the possibilities until something works. For many problems, there are better algorithmic techniques than brute-force, but a brute-force algorithm is often very simple to write code for.

   Use manual multiplication to calculate the square of a number (there is a function to compute exponents, but it can be tricky to use).

2. Write a program that lets the user type in a number from the keyboard and determines if the number is prime or not. You should write a function to determine if a number is prime or not.

   The definition line should look like:

   ```
   bool is_prime(int n)
   ```

3. Write a function to duplicate the random number generator functionality in Python. See the bottom of this page for how to generate random numbers in C++. Specifically, make it so you can generate a random number with an upper bound and a lower bound:

   ```
   int gen_random(int lower, int upper)
   ```

   This should generate a random number x such that `lower <= x <= upper.`

4. Rewrite your guess-the-number game from the first day of class. (Guess-the-number game is when you randomly generate a number and allow the user to keep guessing (with hints like too low, too high) until they guess correctly.) Use the `gen_random` function you wrote in #3.

Random numbers in C++

The built-in random number generator in C++ (called `rand()`) generates a random integer between 0 and a very large number denoted by the built-in constant RAND_MAX. Unlike Python, there is no built-in function to generate a random number between a specific upper and lower bound. Therefore, the most common way to accomplish this is to take the remainder of the return value of `rand()` with the upper bound and add a constant as follows:

```
int v1 = rand() % 100;          // v1 is in the range 0 to 99
int v2 = rand() % 100 + 1;      // v2 is in the range 1 to 100
int v3 = rand() % 30 + 1985;    // v3 is in the range 1985-2014
```

A further complication arises because `rand()` always generates the same sequence of random numbers every time you run your program (unlike Python). (This is actually a useful feature in situations when you need to debug a program that uses random numbers and you want the program to act exactly the same way each time you run it.) To get around this, put the following line of code at the beginning of `main()`:

```
srand(time(0))
```

This "seeds" the random number generator with a new value (based on the current time) each time your program runs, so you get new random numbers each time.

These functions live in the `<cstdlib>` and `<ctime>` libraries, so you should `#include` both of those files at the top of your code when you generate random numbers.