CS142 Lab Day – Practice with loops and functions

1. (skip if already done) Write a function called sumrange that sums up a range of numbers, given an upper and lower bound. The math should include the lower number, but not the upper number. Example: sumrange(1, 5) returns 10 (1+2+3+4).

   ```
   int sumrange(int lower, int upper)
   ```

2. (skip if already done) It is possible for a right triangle to have sides that are all integers (whole numbers). A set of three integer values for the sides of a right triangle is called a Pythagorean triple, such as (3, 4, 5). 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.

   Use manual multiplication to calculate the square of a number, rather than pow, and do not use the square root function (sqrt) either. The reason for this is C++ cannot represent some floating-point numbers exactly inside a computer (just like we can't write down 1/3 exactly as a decimal), so if we compare two floating-point numbers for equality in C++, it's possible they will come back as "not equal" even if they should be equal. In other words, floating-point math sometimes accumulates *round-off errors*. Integer math (assuming the integers can fit in the data type you use) is always exact, so this doesn't happen.

3. Write a function to determine if a number is prime or not. Recall that an integer is prime if it has no other factors other than itself and 1. Return true if the number is prime, false if not. Hint: use a loop.

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

4. Write a function called gen_random that works like the Python random number generator, with an upper and lower bound. The number generated should be in that range, inclusive.

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

5. Write a program that lets the user type in a positive integer from the keyboard. The program should print out the pseudo-Roman numeral equivalent of the number. I say "pseudo" because we will simplify Roman numerals a bit by getting rid of the subtraction rules for Roman numerals. For example, normally 9 is written as IX = 10 – 1, but your program can print VIIII.

   In Roman numerals, M = 1000, D = 500, C = 100, L = 50, X = 10, V = 5, and I = 1.

   Hint: Use a loop that runs until the user's number becomes equal to zero. Inside the loop, write if statements that test how big the number is. If the number is bigger than or equal to one of the exact Roman numerals above, print that numeral, subtract the value from the user's number, and loop again.

   Challenge: make this print out "true" Roman numerals; e.g., for 9 it should print IX, not VIIII. Try to find an algorithm for this on your own, but I have a hint if you really want it.

6. Let the user enter a positive integer from the keyboard, called *n*. Print a triangle, made of asterisks, with base and height of n like this (for *n* = 4):

   ```
   *
   * *
   * * *
   * * * *
   ```

   Then amend your program so it prints the other three variations of this triangle, with the right-angle in the other corners. (These triangles don't all have to be on the same lines as shown below; they can be below each other.)

   ```
   * * * *                         *                 * * * *
   * * *                         * *                   * * *
   * *                         * * *                     * *
   *                         * * * *                       *
   ```