CS 142 Loop Problems

1. Write a program that approximates the value of pi (3.14159…) using the formula

   $\pi = 4 - (4/3) + (4/5) - (4/7) + (4/9) - (4/11) + \ldots$

   Ask the user how many terms of the formula they want to use to approximate pi. For instance, if they enter 6, you would use the six terms above, up to (4/11). Then use a for loop to calculate the approximate value and print it.

   By default, C++ rounds all floating-point numbers, *when printed*, to 6 significant figures, even if in memory they are stored with higher precision. To print a different number of decimal places, use this code:

   ```
   double p = 8.0/3.0;
   cout << p << endl;                                          // prints 2.66667
   cout << std::fixed << setprecision(2) << p << endl;         // prints 2.67
   cout << std::fixed << setprecision(15) << p << endl;        // prints 2.666666666666667
   ```

2. 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 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 in this case because it always returns a floating point number). Do not use the square root function either.

3. Write a program that lets the user type in a number from the keyboard and determines if the number is prime or not.

4. 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.

5. 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:

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