For your first laboratory exercise, we have provided a Java program that computes the square root of a positive floating point number using the Newton-Raphson Method. Run the code in Netbeans and observe its behavior. Your task is to convert this program into a C++ program that produces the same behavior. Make sure your C++ solution maintains the same basic structure (i.e. constants, variables, I/O format, and functions) as the Java version of the code. When you are satisfied that your program works correctly, ask your lab instructor to review and test your solution.

The Newton-Raphson Method (NRM) is an iterative technique for finding the square root of a positive number.

1. We can derive a recurrence relation for finding the square root using the NRM as follows:
   Let \( x = \sqrt{c} \rightarrow x^2 = c \rightarrow x^2 - c = 0 \rightarrow f(x) = x^2 - c \)

2. The NRM defines: \( x_{n+1} = x_n - \left( \frac{f(x_n)}{f'(x_n)} \right) \) where \( f'(x_n) \) is the derivative of the function \( f(x_n) \). 

3. The derivative of \( f(x_n) = (x_n^2 - c) \) is simply \( f'(x_n) = 2x_n \).

4. Substituting \( 2x_n \) for \( f'(x_n) \) in the definition of the NRM produces the recurrence relation:
   \( x_{n+1} = x_n - \left( \frac{f(x_n)}{2x_n} \right) \)

5. Since \( f(x_n) = (x_n^2 - c) \) we can rewrite this recurrence as: \( x_{n+1} = x_n - \left( \frac{(x_n^2 - c)}{2x_n} \right) \)

6. Regrouping we end up with: \( x_{n+1} = 0.5 \times \left( x_n + \left( \frac{c}{x_n} \right) \right) \)

For example, to compute the square root of a number like 36 (e.g. \( c = 36 \)) we apply the following steps:

Guess an initial value for \( x_1 \) (any positive number will work even 1). Let’s use a value of 18 as a starting point.

\[
\begin{align*}
x_1 & = 18.000000 \\
x_2 & = 0.5 \times \left( x_1 + \left( \frac{c}{x_1} \right) \right) = 0.5 \times \left( 18.000000 + \left( \frac{36.000000}{18.000000} \right) \right) = 10.000000 \\
x_3 & = 0.5 \times \left( x_2 + \left( \frac{c}{x_2} \right) \right) = 0.5 \times \left( 10.000000 + \left( \frac{36.000000}{10.000000} \right) \right) = 6.800000 \\
x_4 & = 0.5 \times \left( x_3 + \left( \frac{c}{x_3} \right) \right) = 0.5 \times \left( 6.800000 + \left( \frac{36.000000}{6.800000} \right) \right) = 6.047059 \\
x_5 & = 0.5 \times \left( x_4 + \left( \frac{c}{x_4} \right) \right) = 0.5 \times \left( 6.047059 + \left( \frac{36.000000}{6.047059} \right) \right) = 6.000183 \\
x_6 & = 0.5 \times \left( x_5 + \left( \frac{c}{x_5} \right) \right) = 0.5 \times \left( 6.000183 + \left( \frac{36.000000}{6.000183} \right) \right) = 6.000000
\end{align*}
\]

We have to decide when to stop. A simple approach is to keep going until \( |c - (x_{n+1} \times x_{n+1})| < \epsilon \) where epsilon is a small value that we pick that determines how much error we will tolerate in our approximation. If our epsilon is 0.00001, we would stop with \( x_6 = 6.000000 \).

In summary, to find the square root of a positive number using the NRM, the main program should:

- prompt the user for a positive value (you will find the square root of this value)
- call the NRM square root approximation function passing the value provided by the user
- displays the value and square root of the value
The NRM approximation function should:

- accept the parameter sent from the main program (value you will find the square of)
- computes the square root of the value using NRM (stop when the error is < epsilon)
- return the square root of the value to the calling function

Sample input-output:

Enter value (>0) whose square root will be computed:
36
sqrt( 36.000000 ) = 18.000000
sqrt( 36.000000 ) = 0.5 * ( 18.000000 + ( 36.000000 / 18.000000 ) ) = 10.000000
sqrt( 36.000000 ) = 0.5 * ( 10.000000 + ( 36.000000 / 10.000000 ) ) = 6.800000
sqrt( 36.000000 ) = 0.5 * ( 6.800000 + ( 36.000000 / 6.800000 ) ) = 6.047059
sqrt( 36.000000 ) = 0.5 * ( 6.047059 + ( 36.000000 / 6.047059 ) ) = 6.00183
sqrt( 36.000000 ) = 0.5 * ( 6.00183 + ( 36.000000 / 6.00183 ) ) = 6.000000

sqrt( 36.000000 ) = 6.000000