1. **Introduction**

The purpose of this laboratory is to develop a program that calculates the n-th root of a floating-point number (entered as an argument on the command line) using the so-called Newton-Raphson root-finding algorithm. The program demonstrates the use of a function to solve an analytical (i.e., math-type) problem using either iteration or recursion. As for all laboratory assignments in the course, emphasis is on doing design before coding and applying coding style standards such that correct and maintainable programs are developed quickly.

2. **Before You Start**

How to find the roots of a floating point number

The determination of roots is a very common task. For example, the square root of A is actually the solution, \[ y = \sqrt{A}, \] or \[ y^2 - A = 0. \] The mathematicians Isaac Newton and Joseph Raphson figured out an iterative solution for the general case \[ y = \sqrt[n]{A} \] which is \[ y_{i+1} = \frac{1}{n} \left[ (n-1)y_i + \frac{A}{y_i^{n-1}} \right]. \] A good starting point for \( y \) is \[ y_0 = \frac{A}{n}. \]

Let's see how this works. Assume we want the square root of 5.0, which we know is 2.36067977 (on Hal Carter’s Casio calculator). Then

\[
y_0 = \frac{5.0}{2.0} = 2.5
\]

\[
y_1 = \frac{1}{2} \left[ 2.5 + \frac{5.0}{2.5} \right] = 2.25 \quad y_1 \text{ differs from } y_0 \text{ by 0.25}
\]

\[
y_2 = \frac{1}{2} \left[ 2.25 + \frac{5.0}{2.25} \right] = 2.23611 \quad y_2 \text{ differs from } y_1 \text{ by 0.013889}
\]

\[
y_3 = \frac{1}{2} \left[ 2.23611 + \frac{5.0}{2.23611} \right] = 2.23607 \quad y_3 \text{ differs from } y_2 \text{ by 0.000041601}
\]

As you can see the error (i.e., how much the solution at each iteration differs from the previous iteration result) get smaller quite rapidly.

When do we stop iterating? Let's assume we'll stop when the result of two iterations is smaller than some tiny value, say, 0.000000001. In other words, with this stopping value we are really saying we want to calculate the nth root to at least 10^-9 accuracy. Note that the Newton-Raphson method for finding roots is very efficient. The number of iterations is approximately \( n/2 \) where \( n \) is the number of decimal digits required to meet the accuracy.

**Task 1.** Create a program that displays the n-th root (i.e., square root, cube root, 4th-root, etc.) of a positive floating-point number, \( A \), with at least precision \( p \) (number of digits to the right of the decimal point). \( N, A, \) and \( p \) shall be input as arguments on the execution command line. The format of the input line is: \( N A p \) where \( N \) is the root (>0), \( A \) is the floating point number for which the n-th root is wanted, and \( p \) is the desired precision of the answer. For example, if we want the cube root of 17.4683 calculated to a precision of six digits, the input is: \( 3 \ 17.4683 \ 6 \).
The format of your program displayed output is:

\[
\text{The <name> root of <A> is <answer>}
\]

where <name> is first if \( n \) is 1,

\[
\text{square} \quad \text{if} \quad n = 2,
\]

\[
\text{cube} \quad \text{if} \quad n = 3,
\]

\[
\text{nth} \quad \text{if} \quad n \geq 4.
\]

\( <A> \) is the number for which the root is desired; input on the command line.

\( <answer> \) is the root of \( A \) displayed to at least the precision \( p \). If \( A \) is negative and \( n \) is odd, display \( <answer> \) with an “i” at the end of number to represent an imaginary value.

Example Output: The cube root of 17.4683 is 2.594678

Implementation requirements:

Calculate the \( n \)-th root in a function that accepts \( N, A, \) and \( p \), and returns the root for the main routine to display. Use iteration to solve for the root.

Submission requirements

Submit all code files for this task that implement a complete and correct program (along with an image of the display window of submission is in hardcopy form) to the instructor.

Task 2. In the program you created in Task 1, replace the \( n \)-th-root function with one that uses recursion to calculate the \( n \)-th root. Use the same function parameters used in Task 1, and the same iterative equation for \( y_{i+1} \), but use recursion rather than iteration to solve for the root.

Submission requirements

Submit all code files for this task that implement a complete and correct program (along with an image of the display window of submission is in hardcopy form) to the instructor. If instead your section uses an electronic archive to submit programs for grading, please submit your source files to the archive.