16.216: ECE Application Programming

Fall 2013

Programming Assignment #4: Iterative Algorithms

Due **Monday, 10/7/13**, 11:59:59 PM

1. Introduction

This program will use an iterative algorithm—an algorithm that runs until a given condition is met—to approximate the nth root of a given value. You will also use loops to ensure your application runs until the user explicitly ends the program.

Please note that this operation could be implemented using functions from the C math library. **However, you may not use functions from <math.h> in your solution—any use of a function from <math.h> will result in a 40 point deduction.**

2. Deliverables

Submit your source file directly to Dr. Geiger (Michael_Geiger@uml.edu) as an e-mail attachment. Ensure your source file name is ***prog4_root.c***. You should submit only the .c file. Failure to meet this specification will reduce your grade, as described in the program grading guidelines.

3. Specifications

Input: Your program will repeatedly prompt the user to enter a pair of values separated by a single space: `A n`. If user input is incorrectly formatted, repeat the prompt. The program will then compute the nth root of A, $\sqrt[n]{A}$. Note that:

- A must be a positive real number.
- n must be an integer greater than or equal to 2.
- If either of the above conditions is not met, print an error message and then proceed to the question described below.

Once complete, ask the user if he or she would like to calculate another root. If the user enters `'Y'` or `'y'`, the program should return to its initial prompt. If the user enters `'N'` or `'n'`, the program should exit. Otherwise, print an error message and repeat the question.

Output: Assuming there are no errors, your program should evaluate the inputs and print the values of A and n, as well as the root $\sqrt[n]{A}$. Non-integer values should be printed with two decimal places. Sample input/output pairs are shown below, with the user input underlined:

- `Enter real number and integer (A n): 125 3`
  `Given A = 125.00 and n = 3, root = 5.00`
- `Enter real number and integer (A n): 0.1 10`
  `Given A = 0.10 and n = 10, root = 0.79`
- `Enter real number and integer (A n): 16.216 2`
  `Given A = 16.22 and n = 2, root = 4.03`
- `Enter real number and integer (A n): 65536 4`
  `Given A = 65536.00 and n = 4, root = 16.00`

See Section 5 for additional test cases.

Error checking: As noted above, your program should print an error under any of the following conditions:

- The user enters improperly formatted input values.
- The value of A is negative.
- The value of n is less than 2.
- The user responds to the question about calculating another root with a character other than `'Y'`, `'y'`, `'N'`, or `'n'`.

4. Formulating a solution—hints, tips, etc.

Calculating the nth root: The general method you can use for finding a root is as follows (reference: http://en.wikipedia.org/wiki/Nth_root_algorithm):

- Choose an initial guess, x_0—1 works well as an initial value.
- Iteratively calculate each new guess using the formula:

$$x_{k+1} = \frac{1}{n}\left[(n-1)x_k + \frac{A}{x_k^{n-1}}\right]$$

  Note that A and n are the user input values. I recommend using two variables to store these values:

  1. x_k = result from the previous iteration (should start at 1)
  2. x_{k+1} = result from current iteration

- Stop iterating when the desired precision is reached (when the difference between x_k and x_{k+1} is sufficiently small).
  - I recommend checking that the absolute value of the difference between the two values is < 0.000001.
  - You must check the absolute value because $x_{k+1} – x_k$ may be negative. However, you can't use the built-in absolute value function—find your own method!

Note that if A is 0, your result is 0.

Reading character input: In most cases, `scanf()` will skip whitespace (spaces, tabs, newlines) when reading input. Remember that the exception to that rule comes when using the %c format specifier, which usually reads the next character in the input stream—space or otherwise. Given the following input:

```
5 3
X
```

Say you have the following code, assuming a and b are `int`s and c is a `char`:

```
scanf("%d %d", &a, &b);
scanf("%c", &c);
```

a and b will be 5 and 3, as expected; c, however, will hold the newline character, since that is the first input character after the integer 3. To avoid this problem, you can put a space in your format string, which will cause `scanf()` to skip whitespace characters and read the first character that follows the whitespace. Replace the second line above with:

```
scanf(" %c", &c);
```

Note that newlines in your input may not be obvious—you may enter values, print outputs based on those values, and then prompt for another input value.

5. Test Cases

Your output should match these test cases exactly for the given input values. I will use these test cases in grading of your lab, but will also generate additional cases that will not be publicly available. Note that these test cases may not cover all possible program outcomes. You should create your own tests to help debug your code and ensure proper operation for all possible inputs.