EECE.2160: ECE Application Programming
Spring 2017

Programming Assignment #4: Iterative Algorithms
Due **Wednesday, 2/22/17**, 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 easily 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 by uploading it directly to your Dropbox folder. 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 repeat the prompt asking the user to enter a pair of values.

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 a separate error message for each 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'`.

Each error should cause the program to repeat the prompt to which the user responded with invalid input. Therefore, your program should handle the first three errors by again prompting the user to enter two values, and it should handle the fourth error by repeating the prompt that asks if the user would like to calculate another root.

Input validation (handling input errors and repeating the appropriate prompt if any errors occur) was discussed during the second in-class programming exercise (Lecture 12). I encourage you to look at that lecture for a detailed discussion of this topic.

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

Program design: This assignment is significantly more complex than the first three because the use of loops—in particular, nested loops (loops inside other loops)—makes designing your program more difficult. In particular, you have to determine what code needs to be repeated and under what conditions you repeat that code.

One common method is to break the program into smaller pieces, designing each piece—as well as how they fit together—separately. For example, in Lecture 10 (PE2), we first looked at the overall program flow. In Lecture 12, we'll continue by designing a "block" to strictly handle input validation, as well as blocks for computing n! and 2^n.

The blocks that make up Program 4 are listed below. Note that each block may require one or more loops, and that all of the blocks will need to go inside a larger loop that repeats the entire program when appropriate.

I'd suggest implementing and testing each one individually, then putting them together by adding one new block at a time, ensuring that the new block works appropriately with what you already have.

- Read and validate A and n: Prompt for and read these two values, test for all possible errors, and repeat this block until there are no input errors.

- Calculate the nth root: The details of the iterative algorithm you'll use are below.

- Ask user to repeat: Ask the user if he or she wants to calculate another root. If the user enters anything other than `'Y'`, `'y'`, `'N'`, or `'n'`, repeat the question.

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 additional variables to help you implement the equation:

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

  Also, you can't use the built-in `pow()` function for x_k^{n-1}--you have to write your own version. (You may want to wait until we discuss for loops to handle this part.)

- Stop iterating when the desired precision is reached (when the difference between x_k and x_{k+1} is sufficiently small).

  o I recommend checking that the absolute value of the difference between the two values is > 0.000001—if that's true, keep iterating.

  o You must check the absolute value because $x_{k+1} - x_k$ may be negative. However, you <u>can't</u> use the built-in `fabs()` function—find your own method!

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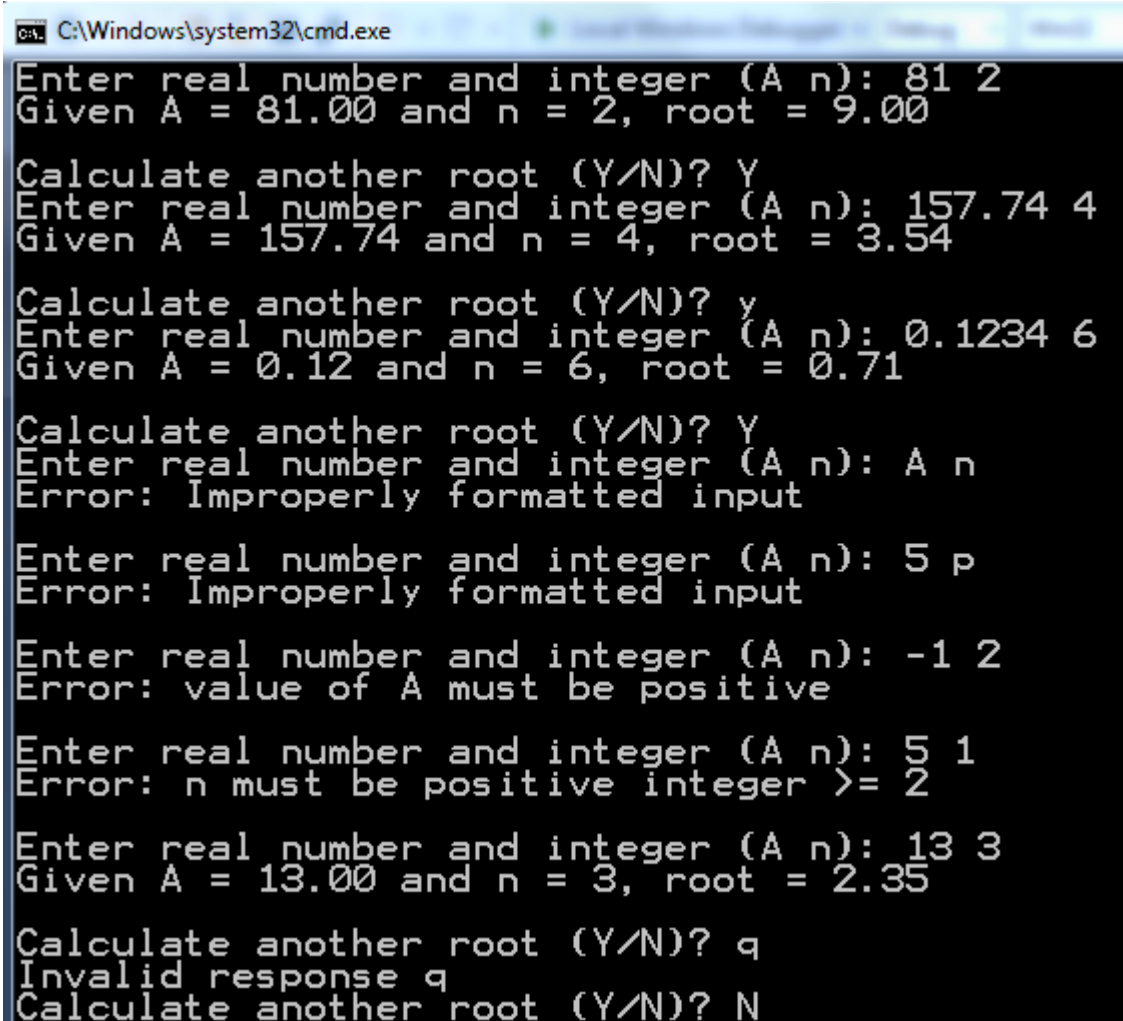
```
scanf("%d %d", &a, &b);
scanf("%c", &c);
```

`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.

```
C:\Windows\system32\cmd.exe

Enter real number and integer (A n): 81 2
Given A = 81.00 and n = 2, root = 9.00

Calculate another root (Y/N)? Y
Enter real number and integer (A n): 157.74 4
Given A = 157.74 and n = 4, root = 3.54

Calculate another root (Y/N)? y
Enter real number and integer (A n): 0.1234 6
Given A = 0.12 and n = 6, root = 0.71

Calculate another root (Y/N)? Y
Enter real number and integer (A n): A n
Error: Improperly formatted input

Enter real number and integer (A n): 5 p
Error: Improperly formatted input

Enter real number and integer (A n): -1 2
Error: value of A must be positive

Enter real number and integer (A n): 5 1
Error: n must be positive integer >= 2

Enter real number and integer (A n): 13 3
Given A = 13.00 and n = 3, root = 2.35

Calculate another root (Y/N)? q
Invalid response q
Calculate another root (Y/N)? N
```