

16.216: ECE Application Programming

Summer 2012

Programming Assignment #5: Programming with Loops

Due **Tuesday, 7/31/12**, 11:59:59 PM

1. Introduction

This program will give you experience working with loops. You will write a program that is similar to your simple calculator from Program 4. However, this program will appropriately handle errors without exiting. In addition, you will implement two new mathematical operations using loops.

Please note that both of these operations—exponentiation and root-finding—could be implemented using functions from the C math library. **However, you may not use functions from `<math.h>` in your solution. Each operation that uses a function from `<math.h>` will result in a deduction of 20 points; you will lose 40 points if both your exponentiation and root-finding solutions use functions from this library.**

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 ***prog5_loops.c***. You should submit only the .c file. Failure to meet this specification will reduce your grade, as described in the program grading guidelines.

This assignment contains the following extra credit sections, which are subject to the guidelines recently posted on the course website:

- (10 points) One of the following additional functions, all of which can be implemented using iterative methods:
 - Base 10 logarithm (A L 10): $\log(A)$
 - Base 2 logarithm (A L 2): $\log_2(A)$
 - Natural logarithm (A N x): $\ln(A)$
 - Note that “x” is an unspecified number; its value can be ignored if you choose to implement this operation.

3. Specifications

Input: Your program will prompt the user to enter an expression of form $A \text{ op } n$, where A is a real number, n is an integer, and op is one of the following operators, described in Section 4: \wedge (exponentiation), R , r (nth root). Examples:

- $-2 \wedge 3$
- $5.2 \wedge -1$
- $9 \text{ R } 2$
- $13.7 \text{ r } 4$

After reading the expression and displaying its result, your program should ask the user if he/she wants to enter another expression:

- If the user enters 'Y' or 'y', return to the start of the program and read a new expression.
- If the user enters 'N' or 'n', exit the program.
- If the user enters anything else, print an error and repeat the question.

Output: Given a valid expression, your program should calculate the result and reprint the entire expression as well as its result, using the default precision (6 places). The expressions listed above will produce the following output:

- $-2.000000 \wedge 3.000000 = -8.000000$
- $5.200000 \wedge 1.000000 = 5.200000$
- $9.000000 \text{ R } 2.000000 = 3.000000$
- $13.700000 \text{ r } 4.000000 = 1.923890$

See Section 5: Test Cases for more sample program runs.

Error checking: Unlike previous assignments, no error should cause your program to exit. Your program should handle the following error conditions:

- Any of the inputs are incorrectly formatted and therefore cannot be read correctly using `scanf()`.
 - Print an error message and repeat the prompt asking the user to enter an expression.
 - Remember, you may need to clear the remainder of the line to ensure the new expression is read correctly.
- The operator entered is not a valid operator.
 - Print an error message, and then ask if the user wants to enter another expression.
- For the nth root operation, the first value is negative and/or the second operand is less than 2.
 - Print an error message, and then ask if the user wants to enter another expression.

4. Designing a solution

Detailed operation discussion: Your operations should use the following commands and conditions:

- Exponentiation (A^n): This command uses \wedge as the operator.
 - Note that n may be positive, zero, or negative—you'll handle each of these three cases differently!
 - Examples:
 - $9 \wedge 2 = 9^2 = 81$
 - $4.5 \wedge 4 = 4.5^4 = 410.0625$

- Root finding ($\sqrt[n]{A}$): This command uses either 'R' or 'r' as the operator.
 - 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:

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

I recommend using two variables to store these values:

- x_k = result from the previous iteration
- x_{k+1} = result from current iteration
- Stop iteration 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!
- This command has the following special cases/error conditions:
 - If A (the first operand) is 0, your result is 0.
 - Otherwise, A must be positive for the algorithm to work—print an error if A is negative.
 - You can assume n (the second operand) is an integer, but it must be ≥ 2 —print an error otherwise.
- Examples:
 - $25 \text{ R } 2 = \text{square root of } 25 = 5.000000$
 - $100 \text{ R } 3 = \text{cube root of } 100 = 4.641589$

4. Designing a solution (cont.)

Clearing input: Some errors can cause your program to get stuck at a single point in the input stream. For example, consider the following code snippet:

```
printf("Enter an integer: ");
n = scanf("%d", &i);
if (n < 1)
    printf("Error: input isn't an integer!");
```

If the user enters a non-numeric input—for example, “A”—the program will print an error message. If you have this code (or something similar) inside a loop, that loop will run forever—your input stream is always going to hold the character A, and your `scanf()` call will never successfully read that value.

To avoid such problems, you need to clear all incorrect input when an error occurs. The code below performs this operation—assuming you have a character variable `junk`, it uses that variable to read all characters left in the line:

```
do {
    scanf("%c", &junk);
} while (junk != '\n');
```

Reading character input: In most cases, `scanf()` will skip whitespace (spaces, tabs, newlines) when reading input. The exception to that rule comes when using the `%c` format specifier, which simply 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 `ints` 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 newline in your format string—replace the second line above with:

```
scanf("\n%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.

4. Designing a solution (cont.)

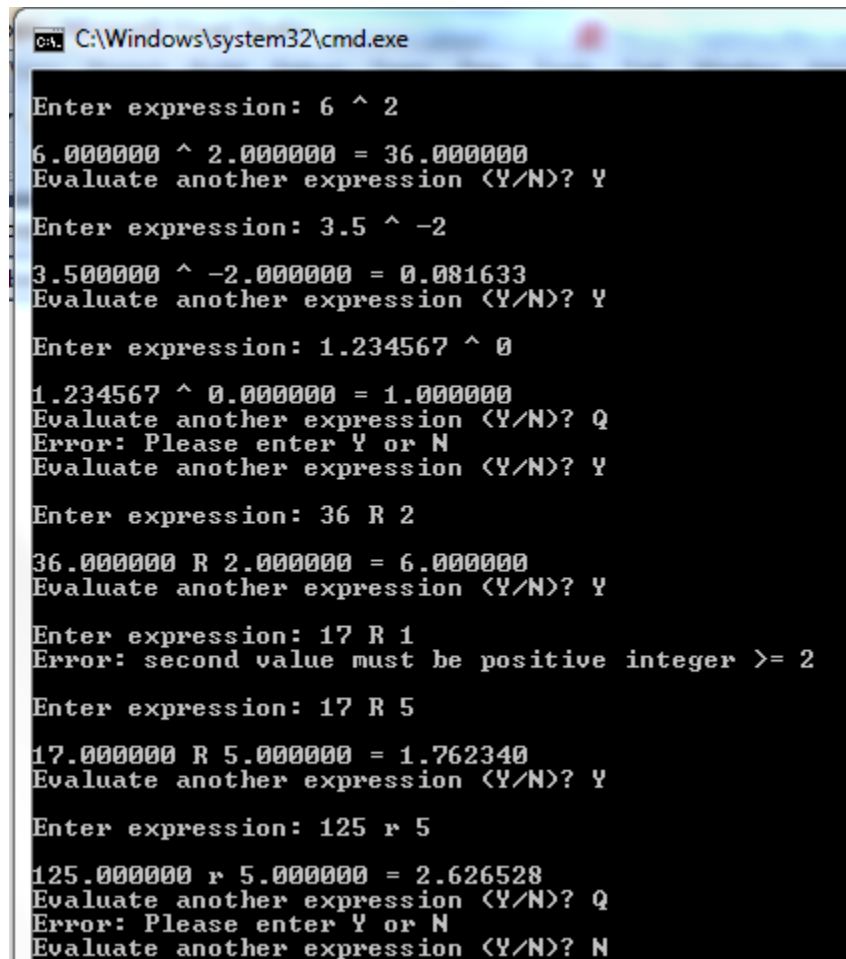
Extra credit: You can earn up to 10 extra points on this assignment by implementing only one of the following operations:

- Base 10 logarithm (A L 10): $\log(A)$. Examples include:
 - $100 \text{ L } 10 = \log(100) = 2$
 - $250 \text{ L } 10 = \log(250) = 2.397940$
- Base 2 logarithm (A L 2): $\log_2(A)$. Examples include:
 - $64 \text{ L } 2 = \log_2(64) = 6$
 - $192 \text{ L } 2 = \log_2(192) = 7.584963$
- Natural logarithm (A N x): $\ln(A)$. Note that “x” is an unspecified number; its value can be ignored if you choose to implement this operation.
Examples include:
 - $100 \text{ N } 1 = \ln(100) = 4.605710$
 - $250 \text{ N } 13 = \ln(250) = 5.521461$

For each operation, you must choose an appropriate iterative method and print the result using the default precision, as shown for the other functions. You may use any code that runs correctly, including material we have not yet covered, **with the exception of functions from the `<math.h>` library.**

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 expression: 6 ^ 2
6.000000 ^ 2.000000 = 36.000000
Evaluate another expression (Y/N)? Y
Enter expression: 3.5 ^ -2
3.500000 ^ -2.000000 = 0.081633
Evaluate another expression (Y/N)? Y
Enter expression: 1.234567 ^ 0
1.234567 ^ 0.000000 = 1.000000
Evaluate another expression (Y/N)? Q
Error: Please enter Y or N
Evaluate another expression (Y/N)? Y
Enter expression: 36 R 2
36.000000 R 2.000000 = 6.000000
Evaluate another expression (Y/N)? Y
Enter expression: 17 R 1
Error: second value must be positive integer >= 2
Enter expression: 17 R 5
17.000000 R 5.000000 = 1.762340
Evaluate another expression (Y/N)? Y
Enter expression: 125 r 5
125.000000 r 5.000000 = 2.626528
Evaluate another expression (Y/N)? Q
Error: Please enter Y or N
Evaluate another expression (Y/N)? N
```

Remember, if you are using Visual Studio, to get your program to terminate with a message saying, "Press any key to continue ...", use the **Start Without Debugging** command (press Ctrl + F5) to run your code.