16.216: ECE Application Programming

Spring 2013

Programming Assignment #5: Working with For Loops

Due **Wednesday, 3/6/13**, 11:59:59 PM

1. Introduction

This program will use `for` loops to perform two different operations: calculating a series approximation of a mathematical constant and raising a value to a given exponent. You will also use loops to ensure your application repeatedly reads input values until the user explicitly ends the program.

Please note that both of these operations—approximating *e* and exponentiation—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 approximation of *e* and exponentiation 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_exp.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 single character command. The program may prompt for and read additional values to be used in the operations described below, depending on what command is entered:

- `'E'`, `'e'`: Prompt the user to enter an integer, *n*, then use *n* to approximate the constant *e* by evaluating the series approximation:

$$e \approx 2.718281828459 \approx \sum_{k=0}^{n} \frac{1}{k!} = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!}$$

  Notes:

  - Recall that *n*! = 1 × 2 × 3 × … × *n*. The first two terms of the expansion are 1 because 0! = 1! = 1.
  - *n* should be positive and less than 13, since 13! = 6,227,020,800, which is greater than the maximum possible integer value. So, if *n* is more than 12 or less than 0, print an error message.

Input (continued):

- `'P'`, `'p'`: Prompt the user to enter two numbers, *x* and *n*, then calculate the value of x^n. Notes:
  - *n* must be an integer, but *x* can be any whole number
  - *n* may be positive, zero, or negative—each case is handled differently!
- `'Q'`, `'q'`: Exit the program.

Note that:

- If the user enters any character other than the ones listed above, print an error message.
- If the user enters any character other than `'Q'` or `'q'`, your program should prompt the user to enter a new command after completing the operation specified for the previous command.

Output: Assuming there are no errors, your program should evaluate the inputs in the manner described above and print the output. Sample input/output pairs are shown below, with the user input underlined (Section 4 has additional cases):

- For the `'E'` or `'e'` command, reprint the value of *n* and print the approximate value of *e* using 9 decimal places:

  -
    ```
    Enter value for n (0 <= n <= 12): 3
    With n = 3, e is approximately 2.666666667
    ```

  -
    ```
    Enter value for n (0 <= n <= 12): 8
    With n = 8, e is approximately 2.718278770
    ```

  -
    ```
    Enter value for n (0 <= n <= 12): 12
    With n = 12, e is approximately 2.718281828
    ```

- For the `'P'` or `'p'` command, reprint the values of x and n, then print the result using 3 decimal places.

  -
    ```
    Enter x and n: 7 5
    7.000000 to the power of 5 is 16807.000
    ```

  -
    ```
    Enter x and n: 1.2 3
    1.200000 to the power of 3 is 1.728
    ```

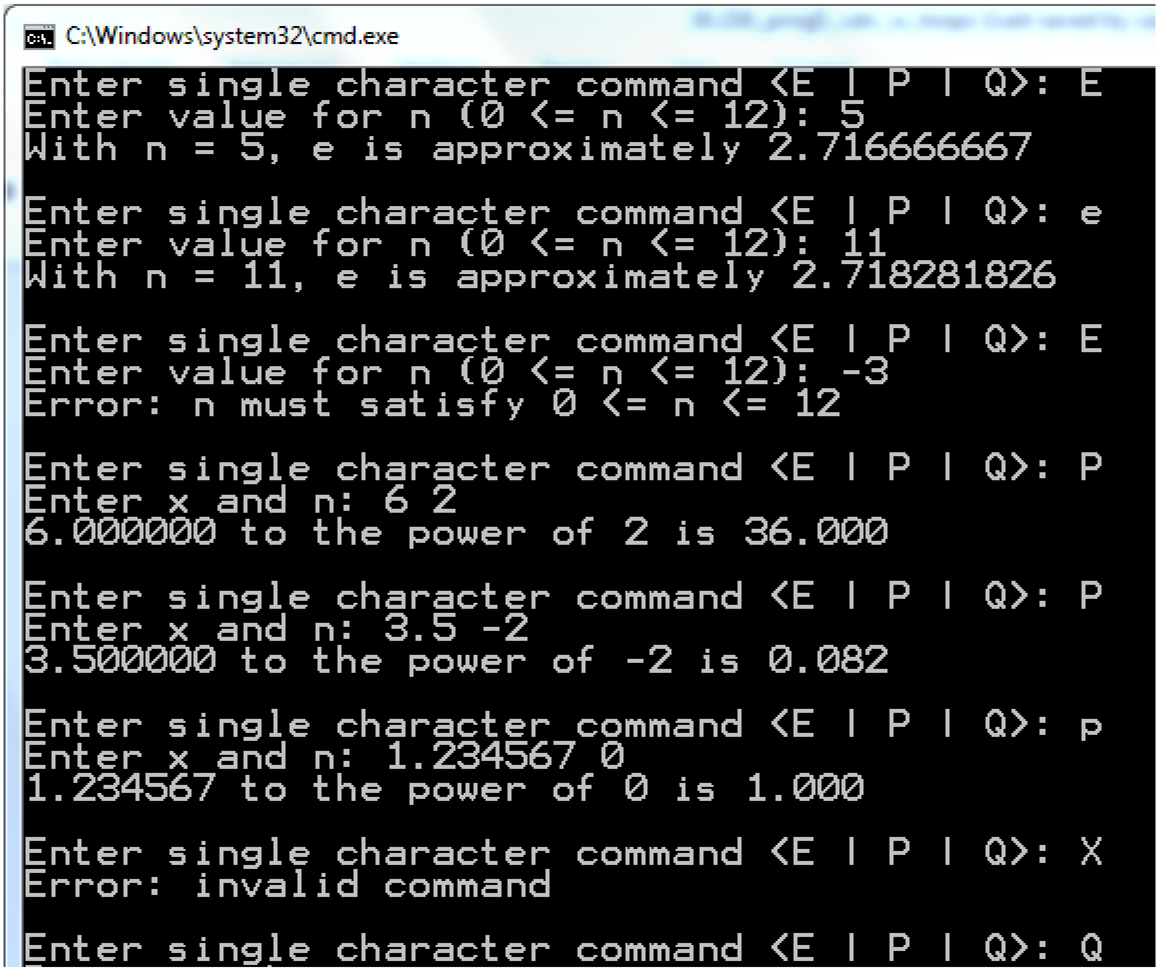
  -
    ```
    Enter x and n: 5 -2
    5.000000 to the power of -2 is 0.040
    ```

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

- The user enters an invalid command.
- For the `'E'` command, the value of n is not between 0 and 12.

4. 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 do 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.



```
Enter single character command <E | P | Q>: E
Enter value for n (0 <= n <= 12): 5
With n = 5, e is approximately 2.716666667

Enter single character command <E | P | Q>: e
Enter value for n (0 <= n <= 12): 11
With n = 11, e is approximately 2.718281826

Enter single character command <E | P | Q>: E
Enter value for n (0 <= n <= 12): -3
Error: n must satisfy 0 <= n <= 12

Enter single character command <E | P | Q>: P
Enter x and n: 6 2
6.000000 to the power of 2 is 36.000

Enter single character command <E | P | Q>: P
Enter x and n: 3.5 -2
3.500000 to the power of -2 is 0.082

Enter single character command <E | P | Q>: p
Enter x and n: 1.234567 0
1.234567 to the power of 0 is 1.000

Enter single character command <E | P | Q>: X
Error: invalid command

Enter single character command <E | P | Q>: Q
```