16.216: ECE Application Programming

Fall 2012

Programming Assignment #4: Working with Loops

Due **Wednesday, 10/17/12**, 11:59:59 PM

1. Introduction

This program will use loops to perform two different operations: reducing a fraction to its lowest terms and calculating a series approximation of a mathematical constant. You will also use loops to ensure your application repeatedly reads input values until the user explicitly ends the program.

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_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.

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:

- `'F'`, `'f'`: Prompt the user to enter a fraction in the form <num>/<den>, then reduce the fraction to its lowest terms using the greatest common divisor of <num> and <den>. Notes:
  - Both the numerator <num> and denominator <den> are integers.
  - The fraction should be entered with no spaces between the two numbers and the `'/'` character that separates them.
  - If the denominator is 0, print an error message.
- `'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.
- `'Q'`, `'q'`: Exit the program.

Input (continued): 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:

- For the `'F'` or `'f'` command, print the original and reduced fractions, and the GCD:
  - `Enter fraction numerator/denominator: 16/80`
    `16/80 reduces to 1/5 (GCD = 16)`
  - `Enter fraction numerator/denominator: 12/99`
    `12/99 reduces to 4/33 (GCD = 3)`
  - `Enter fraction numerator/denominator: 100/40`
    `100/40 reduces to 5/2 (GCD = 20)`
  - `Enter fraction numerator/denominator: 13/15`
    `13/15 reduces to 13/15 (GCD = 1)`
- 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`

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 an invalid command.
- For the `'F'` command, the denominator of the fraction is 0.
- For the `'E'` command, the value of n is not between 0 and 12.

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

Calculating the greatest common divisor (GCD): To reduce a fraction to its lowest terms, you can divide both the numerator and denominator by their GCD. To calculate the GCD, you can use Euclid's algorithm—given two numbers, *x* and *y*, follow the steps below, which are repeated until the GCD is calculated:

1. If *y* is 0, *x* is the GCD.
2. Otherwise, calculate *r*, the remainder of *x* / *y*.
3. Let *x* = *y*, and *y* = *r*.
    - In other words, *x* holds the "old" value of *y*, and the new value of *y* is the remainder from Step 2.
4. Return to Step 1.

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 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 `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 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 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.