# 16.216: ECE Application Programming

Fall 2011

Programming Assignment #5: Improving Your Calculator Due Monday, 10/24/11 Tuesday, 10/25/11, 11:59:59 PM

#### 1. Introduction

This program will give you experience working with loops. You will modify your existing calculator program to provide a more robust interface that does not simply exit when an error occurs. In addition, you will add two new operations that can be implemented using loops.

### 2. Deliverables

Submit your source file using our course site in Blackboard; you should be able to access the site through https://continuinged.uml.edu/login/login.cfm.

Ensure your source file name is **prog5\_loops.c**. You should submit only the .c <u>file</u>. Failure to meet this specification will reduce your grade, as described in the program grading guidelines.

## 3. Specifications

**New feature overview:** Your Program 4 solution should be modified to include the following features, which are discussed in more detail on the following pages:

- The user should have the ability to evaluate multiple expressions in one run of the program.
- The program should not exit when an error occurs. Instead, it should print an error message, and then repeat the appropriate prompt:
  - o If the precision is incorrect, the user should re-enter the precision.
  - o If the expression to be evaluated is incorrectly formatted, the user should re-enter only the expression.
  - If there are any other errors (divide by 0, invalid operator, or issues with the new operations), the user should re-enter both precision and expression.
- You will add two new operations (more details follow on Page 2), both of which will use loops:
  - $\circ$  Exponentiation—raising a number to a given power, i.e.  $x^y$ .
  - o Computing the nth root of a number,  $\sqrt[n]{A}$ .
- Please note that, for these new functions, you may not use any of the functions in the <math.h> library. Doing so will result in a 10 point deduction for each operator—20 points if you use the math library for both.

## 3. Specifications (cont.)

**Detailed discussion of new functions:** The new functions should use the following commands and conditions:

- Exponentiation (a<sup>b</sup>): This command uses either 'P' or 'p' as the operator.
  - You can assume that b (the second operand) is an integer value, even though your program will store it as a float or double.
    - Note that b may be positive, zero, or negative—you'll handle each of these three cases differently!
  - o Examples:
    - 9 P  $2 = 9^2 = 81$
    - $4.5 p 4 = 4.5^4 = 410.0625$
- 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<sub>0</sub>—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 <u>can't</u> 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  $\geq 2$ —print an error otherwise.
- o Examples:
  - 25 R 2 = square root of 25 = 5
  - 100 R 3 = cube root of 100 = 4.641589 (with precision==6)

# 3. Specifications (cont.)

**General interface:** The general interface for the calculator remains the same:

- The user enters a desired precision for the result and operands, followed by a simple arithmetic expression of the form *a op b*, where *a* and *b* are operands and *op* is the operation to be performed.
- Assuming the inputs are correct, your program should calculate the result and print the entire expression, using the precision entered for all values.
- Your program must continue to check for the following four errors and handle them as described on the first page.
  - o Incorrectly formatted input—in other words, the user input values do not match the format specifiers used in your calls to scanf().
  - o Invalid precision—precision must be  $\ge$  0.
  - o Divide by zero
  - Invalid operator

### 4. Designing a solution

**General design:** While there are many valid ways to complete this program, I suggest the following process:

- Modify your original calculator program to allow you to evaluate multiple expressions in a single run of the program.
  - Place a loop around the entire program that exits when the user enters the appropriate character.
  - At the end of this loop, ask the user if he or she would like to evaluate another expression—'Y' allows you to repeat the program, 'N' exits the program, and any other character is an error.
- Modify your program so that errors do not cause the program to exit—instead, the user is prompted again to enter a correct value (or, as described above, to re-enter both precision and expression in some cases).
  - See the hint on the next page on clearing the input, as handling some errors will require you to do so.
- Add the exponentiation operation to your program.
- Add the root finding operation to your program.

## 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!");</pre>
```

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 <code>junk</code>, 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, <code>scanf()</code> 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.

### 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 precision: -1
Error: negative precision
Enter precision: 2
Enter expression: A - B
Enter expression: H - B
Error: incorrectly formatted input
Enter expression: 1 + Z
Error: incorrectly formatted input
Enter expression: 3.5 * 2.0
3.50 * 2.00 = 7.00
Evaluate another expression (Y/N)? Y
Enter precision: 4
Enter expression: 7.2 / 0
Error: cannot divide by 0
Enter precision: 3
Enter expression: 6 P 2
6.000 P 2.000 = 36.000
Evaluate another expression (Y/N)? Q
Error: Please enter Y or N
Evaluate another expression (Y/N)? Y
Enter precision: 4
Enter expression: 1.234 P 3
1.2340 P 3.0000 = 1.8791
Evaluate another expression (Y/N)? Y
Enter precision: 4
Enter expression: 2 P -3
2.0000 P -3.0000 = 0.1250
Evaluate another expression (Y/N)? Y
Enter precision: 2
Enter expression: 49 r 2
49.00 r 2.00 = 7.00
Evaluate another expression (Y/N)? Y
Enter precision: 6
Enter expression: 7.5 r 3
7.500000 r 3.000000 = 1.957434
Evaluate another expression (Y/N)? N
Press any key to continue . . .
```

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.