CCPS 393 - Assignment 1

Write a Unix shell program (i.e. script) called "calc" to simulate a simple calculator.

Key functional requirements:

- Begin by displaying a brief set of instructions on how to use the calculator.
- Prompt the user for an operand followed by an operator and then the second operand.
- Display the two numbers, the operation and the answer (see example sequence below).
- As each subsequent operation will use the previous result, prompt the user for the next operator and operand.
- Implement integer arithmetic only (no need for floating point support).
- Check for division by zero; display warning and continually ask the user to re-enter the divisor.
- At the "Operand" prompt, the user can enter: Any digit, MR, MC, Clear (C), Off (X)
- At the "Operator" prompt, the user can enter: +, -, *, /, MS, M+, MC, Clear (C), Off (X)
- At both prompts (treat as operand or operator): MC, Clear (C), Off (X)
- Error checking: For the operand, assume valid input (no error checking required). For the operator, check for an invalid operator, display a brief informative message, then re-ask the user for input; do not "clear" the transaction.
- Initialize variables appropriately (hint: memory).
- Program (script) must be directly executable (i.e. run with "./calc" NOT "bash calc")
- Do not implement operator precedence, or Reverse Polish Notation, or anything complicated: Think dollar store calculator.
- Work towards a solution which minimizes code duplication.
- Match your code behaviour to the sample dialog below.

Comprehensive section (mandatory):

- Add a section of comments in the file containing your code (The # (hash/numbersign) character introduces a comment in Unix scripts) in which you answer the following questions:
    1. Name two Unix or programming concepts which this assignment helped you learn or become more comfortable.
    2. Describe two difficulties or problems which you encountered in completing the assignment. These could be academic, technical, or non-technical.
- Answer each question in a few sentences.
- See sample comment block below. You can paste this and fill in your own words.

```
# This is a comment in Unix
#
# Comprehensive section:
#
# 1. This assignment helped ...
#
# 2. My two difficulties were ...
#
```

Implement the following operations using the following keys or key sequences:

User input	**Function**	**Required action**
digits	operand entry	accept entry
+	Addition	arithmetic
-	Subtraction	arithmetic
*	Multiplication	arithmetic
/	Division	arithmetic
C	Clear	clear result; keep memory
MS	Memory Store	stores result in memory
M+	Memory Add	adds result to memory
MR	Recall	can be 1st or 2nd operand
MC	Memory Clear	puts zero in memory
X	Off	exit to Unix

Example sequence: (not exhaustive testing)

```
$ ./calc
[ *** User instructions displayed *** ]
Enter operand: 3
  Enter operator: +
Enter operand: 4
     3 + 4 = 7
  Enter operator: -
Enter operand: 12
     7 - 12 = -5
  Enter operator: MS
     -5 -> M
  Enter operator: *
Enter operand: 3
     -5 * 3 = -15
  Enter operator: C
     results cleared
Enter operand: MR
     M -> -5
  Enter operator: *
Enter operand: MR
     M -> -5
     -5 * -5 = 25
  Enter operator: M+
     25 + -5 = 20 -> M
  Enter operator: /
Enter operand: 0
Can't divide by zero.  Please re-enter divisor.
Enter operand: 0
Can't divide by zero.  Please re-enter divisor.
Enter operand: -6
     25 / -6 = -4
  Enter operator: +
Enter operand: C
     results cleared
Enter operand: -9
  Enter operator: k
Sorry, k, is not a valid operator.  Please re-enter operator.
  Enter operator: MC
   0 -> M
  Enter operator: +
Enter operand: 3
   -9 + 3 = -6
  Enter operator: X
Good bye.
$
```

Suggestions on programming in an unfamiliar environment (General comments):

Good	Bad
• write 2 or 3 lines and get that working • add a few more lines • experiment by changing a line or two • gradually add functionality • build confidence	• type in an entire program without ever testing a single line • stumble on syntax error on line 2 • stumble again every few lines with syntax errors • discover that your whole approach was wrong, but you are too heavily invested to re-write • become frustrated and discouraged

Suggestions for getting started on this assignment:

Begin with the looping and control structure. Focus on the control flow and sequencing (operand vs operator, etc.). Initially ignore the arithmetic -- implement that later. Use fake output statements as placeholders (i.e. echo "2 + 3 = 5" with no computation yet).

Marking will be based on:

- Computation – correctness of results
- Logic - how accurately does it simulate a real calculator (control flow)
- Comments/Documentation

Don't be too concerned about the number output format. I am mainly interested in the ability of your program to perform input/output and correct control flow. Pay close attention to the sequence of prompts in the above example.

Assig: calculator marking scheme

Comprehensive section: 2

- mandatory section: must be completed to earn any marks

- demonstrates awareness and reflection

Function and Logic: 8

- how accurately control flow simulates a calculator
- correct control sequencing (e.g. prompts for operand after operator; study after operand 4 in above sample)
- can press MC, Clear, Off anytime even at first prompt (-1 each if not supported)
- correct arithmetic
- checks for division by zero & re-prompt for operand
- MR for both operands
- works for negative numbers

Documentation & Style: 2

- follow the Widget Corporate Style Guide
- good use of variable names ($result, $operand, etc. NOT $a, $b, $c, etc.) (-1)
- answers question WHY? (-1)
- must achieve greater than 4/8 in function and logic for doc & style marks to count (i.e. can't pass if calculator not really working)

Penalties:

- missing or inadequate Comprehensive section - no marks (zero) for the assignment
- control sequencing error (e.g. prompting of operator/operand in wrong order) (-3)
- fail to clean up temp files: -1
- missing user instructions: -1
- script not directly executable -1

Please note: Programs are expected to be **free from syntax errors** and should not "bomb". Programs which generate syntax errors or terminate abnormally will not be eligible for partial credit, even if a few parts "sort-of" work.

Submission Instructions

Please refer to the "Submission Instructions" in the D2L course shell.