Intro to Computer Science
Pep/8 Calculator Project

The purpose of this project is to build a simple calculator program in Pep/8 that supports +, -, *, / and parentheses, as well as the TI feature of retaining the last answer and being able to use it to start off the next calculation.

To complete this project without driving yourself crazy, you will need to use the Pep/8 CALL and RETn Pep/8 instructions to create subprogram abstractions similar to the methods of Alice. See Figure 6.18 in the Pep/8 Help Examples for an example of them. To accomplish Milestone 9 and higher you're going to want to use the Index register to implement a "stack". Get the handout from MrH on how to do this when you are past Milestone 5.

The Milestones for this project are listed below. They are cumulative; for example, Milestone 5 must include all the functionality of Milestones 1-4 except that Milestone 5's Use Answer functionality can be omitted from Milestones 6-10 as noted below.

1	+/- Lines	The user enters a number, then a plus or minus, then another number, and the program outputs the sum. Report an error if the user enters something besides a + or – as the operator. Each entry is on a separate line (the user presses Enter after each one). For example:
		`34`
		`+`
		`2`
		`=36`
		`-2`
		`-`
		`4`
		`=-6`
		`3`
		`*`
		`3`
		`Error: invalid expression`
		Note that this allows you to use DECI and CHARI; it's OK to let DECI terminate the program if the user enters an invalid number.

2	Detect Overflow	Detect and report any two's complement overflow as an error on entry or after an operation. For example:
		`32767`
		`+`
		`2`
		`Error: overflow`
		`32768`
		`Error: overflow`
		`-32768`
		`-`
		`1`
		`Error: overflow`

3	Multiply	Support * for multiply, including positive and negative numbers. Detect and report overflow as an error.
4	Divide	Support / for divide, including positive and negative numbers. This is

		integer division, rounded down; that is, 5/3 = 1. Detect and report division by zero as an error.
5	n op n	From here on, use a single line to enter the complete expression to be evaluated. You must continue to detect and report overflow and division by 0. For this Milestone, the valid expressions are a single digit, followed by a +, -, * or /, and then another single digit. No spaces are allowed. From now on, YOU must detect and report any invalid expression, and allow the user to continue by entering a new expression.
6	Use Answer	Save the result of the last operation, and use it if the user enters an operator first, as a TI-83 does. *Note that this means you are going to have to use CHARI for the first input, and if it's a number, create an integer from the string of characters entered, including detecting overflow and invalid characters! This will also temporarily prevent you from entering a negative number first.*

7	num op num	Allow multi-digit numbers. For example:
		`12*12`
		`=144`

8	+/- numbers	Support a leading + or – with any number to indicate positive or negative. There is no separate "negative" sign like the TI calculator has, so you can't start an expression with a negative number unless the previous answer was a 0. For example:
		`-4+2`
		`=-38`
		`0`
		`=0`
		`-4+2`
		`=-2`

9	multi-op	Allow any number of operators strung together, but simply done in order (ignoring PMDAS). For example:
		`0-4+2`
		`=-2`
		`3+4*-3`
		`=-21`

10	MDAS	Apply the usual order-of-operations rules:
		`3+4*3`
		`=15`
		`2*3+2+4*3`
		`=20`
		This can be done without a stack, but it's probably easier with one, and you're going to need it for the next Milestone anyway.

11	PMDAS	Add parentheses. Any number of parentheses are allowed:
		`((2+3)*2+1)*4`
		`=44`
		`(2)`
		`=2`
		`(((2+3)))`
		`=5`

You should submit only one .pep file for a Milestone.

For full credit for the Milestone, it must:

1. Have a comment copyright notice at the top, claiming your ownership of the code
2. Be organized into major blocks with comments before each that:
   a. Says what the block does (as in "Finds the GCD of two positive integers")
   b. Explains at a high-level how it works (as in "Subtracts the larger number from the smaller one until they're equal")
   c. Lists all sources (web or otherwise), including the specific ideas that you used from each source.
3. Uses the standard tab convention to separate labels, instructions, and comments
4. Doesn't contain significant amounts of duplicated code that could be eliminated with subprograms.