You’ve been given the program and the following description of it:

The program is designed to add two numbers, which you enter. Each number should be one or two digits. The program will echo your entries, and then print the sum. Press <Enter> after each number. To start the program, type ADDER.

Step 1: Start With an Obvious and Simple Test

What you do	What happens
Type ADDER and press <Enter>	The screen blanks. You see a question mark at the top of the screen
Press 2	A 2 appears after the question mark
Press <Enter>	A question mark appears on the next line
Press 3	A 3 appears after the second question mark
Press <Enter>	A 5 appears on the third line. A couple of lines below another question mark appears

```
?     2
?     3
5

?     _
```

The program worked, but do you see any problems with it?

1. Design Error: Nothing shows you what program you’re running. How do you know you’re in the right program?

2. Design Error: There are no onscreen instructions. How do you know what to do? What if you make a typing mistake? Instructions could easily be displayed on the screen.

3. Design Error: How do you stop the program? These instructions should appear onscreen too.

4. Coding Error: The sum (5) isn’t lined up with the other displayed numbers.

Step 2: Design some test cases on “valid” inputs

How many combinations of valid inputs are there?

20 different 1-digit numbers

200 different 2-digit numbers

220 x 220 = 48,400

We tried 2 + 3 as our first test case. What do you think about using 4 + 5 as our next test case?

4 + 5 is so similar to 2 + 3 that we are unlikely to learn anything from 4 + 5 that we didn’t learn from 2 + 3.

OK, what test cases should we use?

Test case	Expected results	Justification
99 + 99	198	Largest pair of numbers you can add
-99 + -99	-198	Smallest pair of numbers you can add
99 + -99	0	Add positive with negative
-99 + 99	0	Add negative with positive
99 + 1	100	Large positive with small positive
99 + -1	98	Large positive with large negative
-99 + 1	-98	Small negative with small positive
-99 + -1	-100	Small negative with large negative
0 + 0	0	Programs often fail on zero
99 + 0	99	
0 + -99	-99	

Step 3: Design some test cases that include use of <Backspace> to correct typing errors

Before we said that there are 48,400 different input combinations. What if users are allowed to use <Backspace> to correct typing errors? How many different input combinations are there now?

Infinite

Try the previous test cases by entering each number, erasing it with <Backspace> and typing it again.

Step 4: Design some test cases that use invalid inputs

What kinds of test cases should we run on invalid inputs?

100 + 100	100 is the smallest invalid positive
-100 + -100	-100 is the largest invalid negative
+9 + +10	Does it allow unary + as well as unary -
<SP><SP>10 + <SP><SP>3	What about spaces?
10<SP><SP> + 3<SP><SP>	
1 * 1 + 2 / 2	Arithmetic operators?
(5) + (-6)	Parentheses?
$10 + $5	Dollar signs?
10% + 5%	Percent signs?
<Enter> + <Enter>	Empty numbers?
1.2 + 5	Floating-point not allowed
A + b	Invalid characters
2A + 2b	Combination of valid and invalid characters
<Ctrl-A> + <Ctrl-B>	Strange characters often cause a crash
<F1> + <Esc>	

How long of a number can be typed in? We expect the program to reject numbers with more than two digits, but what happens if you type in a really long number and hit <Enter>?

1) Program correctly rejects the number
2) Program ignores all but the first two digits of the number (not really correct)
3) Buffer overflow (program does 1 or 2, but crashes later)

What happens if you type in a really long value, possibly including invalid characters, and then use <Backspace> to achieve a valid input? What about <DELETE>, <INSERT>, and the cursor movement keys?

Etc. Etc. Etc.

This example was taken, with modifications, from *Testing Computer Software*, 2nd Ed., by Cem Kaner, et. al.