Homework, Week 7, part B

lightprogramming.org

October 21, 2016

After determining the requirements of the Hangman game, you should create a design for the game. In this case, a first try at writing a requirements specification may include the following tasks. Since writing software is an iterative task, this specification will not be complete, so you will expect to add to it as you discover more about the problem. Here are some requirements you may have written:

- allow the first play to enter a word or phrase to be guessed (we'll call this the *answer*)
- hide the characters in the *answer* so the second player cannot see it
- allow the second player to guess individual characters (we'll call these *guesses*)
- check to see if the guesses appear in the answer
- count the wrong guesses and end the game after six wrong guesses
- end the game if the second player guesses the word

Let's work on the first two items. The Lisp function `read-line` will read a line of text from the console, so that will work fine, but will notice two problems First, you have to treat upper and lower characters the same. Second. it's much more convenient to deal with lists that text. The Lisp function `string-upcase` solves the first problem by converting all characters to upper case. The function `coerce` solves the second by coercing the text to a list. Here is my code.

```
(defun get-answer ()
  (format t "Enter the word or pharse to be guessed: ~%")
  (coerce (string-upcase (read-line)) 'list))
```

Test this by evaluating `(get-answer)`, and then declare and initialize a variable (call it *answer*) by the following: `(setf answer (get-answer))`.

The next problem is to replace the characters with underscores. We could use underscores, but let's use asterisks (*) because they look better. To do this, replace every character with an *, unless the player has already guessed the character. This is easy — write a function that takes a list of characters and replaces the first character with an *, and then calls itself with the `cdr` of the list. Obviously, you will have to also pass an argument consisting of the previous good guesses. Here is my function. It looks more complicated than it is.

```
(defun get-obscure (good-guesses answer)
  (cond
    ((null answer) nil)
    (t
      (cons
        (if
        ;this is the test
          (or ;either a good guess or not an alphabetical character
            (member (car answer) good-guesses)
            (not (alpha-char-p (car answer))))
          ;then
           (car answer)
          ;else
           #\*)
      (get-obscure good-guesses (cdr answer)))))) ;recursive call
```

Read this carefully and notice the logic. This function takes two arguments, both of which are lists, *good-guesses* and *answer*. If the *answer* list is empty, the function returns nil. Otherwise, it does an if-then-else statement. If the next character is in the list of good-guesses or the character is not alphabetical, the function returns that character, else it returns an asterisk, and then calls itself with the remainder of the list. Here is how it works. I turned on `trace` so you can see it.

Line [93] declares a variable named *answer* and initializes it from tne keyboard with the text "common lisp." Notice that these characters are lower case and that the phrase contains a space. Line [94] calls `get-obscure` with no good guesses. Line [95] calls `get-obscure` with the good-guesses list containing an "O" and an "M."

```
[93]> (setf answer (get-answer))
Enter the word or pharse to be guessed:
common lisp
(#\C #\O #\M #\M #\O #\N #\Space #\L #\I #\S #\P)
[94]> (get-obscure () answer)
1. Trace: (GET-OBSCURE 'NIL '(#\C #\O #\M #\M #\O #\N #\Space #\L #\I #\S #\P))
2. Trace: (GET-OBSCURE 'NIL '(#\O #\M #\M #\O #\N #\Space #\L #\I #\S #\P))
3. Trace: (GET-OBSCURE 'NIL '(#\M #\M #\O #\N #\Space #\L #\I #\S #\P))
4. Trace: (GET-OBSCURE 'NIL '(#\M #\O #\N #\Space #\L #\I #\S #\P))
5. Trace: (GET-OBSCURE 'NIL '(#\O #\N #\Space #\L #\I #\S #\P))
6. Trace: (GET-OBSCURE 'NIL '(#\N #\Space #\L #\I #\S #\P))
7. Trace: (GET-OBSCURE 'NIL '(#\Space #\L #\I #\S #\P))
8. Trace: (GET-OBSCURE 'NIL '(#\L #\I #\S #\P))
9. Trace: (GET-OBSCURE 'NIL '(#\I #\S #\P))
10. Trace: (GET-OBSCURE 'NIL '(#\S #\P))
11. Trace: (GET-OBSCURE 'NIL '(#\P))
12. Trace: (GET-OBSCURE 'NIL 'NIL)
12. Trace: GET-OBSCURE ==> NIL
11. Trace: GET-OBSCURE ==> (#\*)
10. Trace: GET-OBSCURE ==> (#\* #\*)
9. Trace: GET-OBSCURE ==> (#\* #\* #\*)
8. Trace: GET-OBSCURE ==> (#\* #\* #\* #\*)
7. Trace: GET-OBSCURE ==> (#\Space #\* #\* #\* #\*)
6. Trace: GET-OBSCURE ==> (#\* #\Space #\* #\* #\* #\*)
5. Trace: GET-OBSCURE ==> (#\* #\* #\Space #\* #\* #\* #\*)
4. Trace: GET-OBSCURE ==> (#\* #\* #\* #\Space #\* #\* #\* #\*)
3. Trace: GET-OBSCURE ==> (#\* #\* #\* #\* #\Space #\* #\* #\* #\*)
2. Trace: GET-OBSCURE ==> (#\* #\* #\* #\* #\* #\Space #\* #\* #\* #\*)
1. Trace: GET-OBSCURE ==> (#\* #\* #\* #\* #\* #\* #\Space #\* #\* #\* #\*)
(#\* #\* #\* #\* #\* #\* #\Space #\* #\* #\* #\*)
[95]> (get-obscure '(#\O #\M) answer)
1. Trace: (GET-OBSCURE '(#\O #\M) '(#\C #\O #\M #\M #\O #\N #\Space #\L #\I #\S #\P))
2. Trace: (GET-OBSCURE '(#\O #\M) '(#\O #\M #\M #\O #\N #\Space #\L #\I #\S #\P))
3. Trace: (GET-OBSCURE '(#\O #\M) '(#\M #\M #\O #\N #\Space #\L #\I #\S #\P))
4. Trace: (GET-OBSCURE '(#\O #\M) '(#\M #\O #\N #\Space #\L #\I #\S #\P))
5. Trace: (GET-OBSCURE '(#\O #\M) '(#\O #\N #\Space #\L #\I #\S #\P))
```

```
6. Trace: (GET-OBSCURE '(#\O #\M) '(#\N #\Space #\L #\I #\S #\P))
7. Trace: (GET-OBSCURE '(#\O #\M) '(#\Space #\L #\I #\S #\P))
8. Trace: (GET-OBSCURE '(#\O #\M) '(#\L #\I #\S #\P))
9. Trace: (GET-OBSCURE '(#\O #\M) '(#\I #\S #\P))
10. Trace: (GET-OBSCURE '(#\O #\M) '(#\S #\P))
11. Trace: (GET-OBSCURE '(#\O #\M) '(#\P))
12. Trace: (GET-OBSCURE '(#\O #\M) 'NIL)
12. Trace: GET-OBSCURE ==> NIL
11. Trace: GET-OBSCURE ==> (#\*)
10. Trace: GET-OBSCURE ==> (#\* #\*)
9. Trace: GET-OBSCURE ==> (#\* #\* #\*)
8. Trace: GET-OBSCURE ==> (#\* #\* #\* #\*)
7. Trace: GET-OBSCURE ==> (#\Space #\* #\* #\* #\*)
6. Trace: GET-OBSCURE ==> (#\* #\Space #\* #\* #\* #\*)
5. Trace: GET-OBSCURE ==> (#\O #\* #\Space #\* #\* #\* #\*)
4. Trace: GET-OBSCURE ==> (#\M #\O #\* #\Space #\* #\* #\* #\*)
3. Trace: GET-OBSCURE ==> (#\M #\M #\O #\* #\Space #\* #\* #\* #\*)
2. Trace: GET-OBSCURE ==> (#\O #\M #\M #\O #\* #\Space #\* #\* #\* #\*)
1. Trace: GET-OBSCURE ==> (#\* #\O #\M #\M #\O #\* #\Space #\* #\* #\* #\*)
(#\* #\O #\M #\M #\O #\* #\Space #\* #\* #\* #\*)
[96]>
```