CSE 341 — Racket Discussion Questions Part 2

The first two questions are additional practice questions on recursion. After that, these questions deal with structs, representing objects, lexical scoping, and macros.

1. Define a recursive function \texttt{flip} that takes a list of booleans and returns a list with \texttt{not} applied to each value. For example, \texttt{(flip '(#t #t #f))} should return \texttt{(#f #f #t)}.

2. Define another version of \texttt{flip} using \texttt{map}.

3. Define a function \texttt{map2} that takes a 2-argument function and two lists. It should return a list of the results of applying the function to corresponding pairs of elements from the two lists. For example, \texttt{(map2 + '(1 2 3) '(10 11 12))} should evaluate to \texttt{(11 13 15)}.

How did you decide to handle the case of lists of different length? Justify your answer.

4. What does this expression evaluate to? Why? (What environment is \texttt{(f 3)} evaluated in? What environment is the body of the lambda evaluated in?)

\begin{verbatim}
(let ([x 2])
  (let ([f (lambda (n) (+ x n))])
    (let ([x 17])
      (f 3))))
\end{verbatim}

5. What does this expression evaluate to? Why?

\begin{verbatim}
(define (addN n)
  (lambda (m) (+ m n)))

(let* ([m 10]
       [n 20]
       [addit (addN 3)])
  (addit 100))
\end{verbatim}

6. What is the result of evaluating this expression? Why?

\begin{verbatim}
(let ([f (lambda () (/ 1 0))]
    [x (+ 3 4)]
  (+ x x))
\end{verbatim}

7. Define a struct called \texttt{point3d} that represents 3D points. Create a point \texttt{p} at the origin; change its \texttt{z} value to be \texttt{10}; and print it out. It should print as \texttt{(point3d 0 0 10)}.

8. Define a \texttt{make-cell} function that returns a simulated instance of a cell with a single field \texttt{value}, which should be hidden (using lexical scoping). The cell should provide “methods” for \texttt{get-value} and \texttt{set-value!}. Follow the bank account example in doing this. The value should start out as null.

9. Similarly but with more bells and whistles... define a \texttt{make-point} function that returns a simulated instance of point with \texttt{x} and \texttt{y} fields, which should be hidden (using lexical scoping). The point should provide “methods” for \texttt{get-x}, \texttt{get-y}, \texttt{set-x!}, \texttt{set-y!}, and \texttt{print-point}. Follow the bank account example in doing this. The fields should start out as \texttt{0}.

10. Define a Racket macro \texttt{and2} that is a 2-argument version of \texttt{and}. Hint: the value of the \texttt{and} expression in Racket is the value of the \texttt{last} subexpression if all of them are something other than \texttt{#f}. The \texttt{and2} macro should work the same, so \texttt{(and2 #t "squid")} should evaluate to \texttt{"squid"}.

11. Define a Racket macro \texttt{grump} that takes one argument and always returns \texttt{"no"}. The argument should not be evaluated.