Objective:
Write macros and read-macros in Common Lisp, using backquote, gensym and other common techniques.

What to do:
1. Using the struct definition of a card in cards3.lsp, write a read macro that makes a card structure out of a list of suit and rank. Use the #C dispatch macro character.
   Example:
   \texttt{>\#C(S J)}
   \texttt{#S(CARD :SUIT S :RANK J)}
2. Using the struct definition of an iterator and the definition of \texttt{list-iterator} below, define a macro \texttt{with-list-iterator} that works like \texttt{with-open-file} but using a list iterator. The code should define local no-argument functions \texttt{next} and \texttt{has-next} that call the appropriate functions in the defined iterator.
   Example:
   \texttt{(with-list-iterator (i (a b c d e f))}
   \texttt{(next)}
   \texttt{(if (has-next) (format t "~A~" (next)))}
   should print B.
3. Define a macro \texttt{doiter} that is comparable to \texttt{dolist} except that it moves through an iteration. The macro should expand into a \texttt{do} loop.
   Example:
   \texttt{(doiter}
   \texttt{(v (list-iterator '(a b c)))}
   \texttt{(format t "~A~" v))}
   Should print out A B C on separate lines.
   \texttt{(with-list-iterator (i '(99 2 3 4))}
   \texttt{(let ((ans 1))}
   \texttt{(next)}
   \texttt{(doiter (v i ans)}
   \texttt{(setf ans (* ans v)))}
   Should return 24

(Extra credit) Write \texttt{doiter} to use a local recursive function defined with \texttt{labels}, rather than a \texttt{do} loop. Note that you must avoid symbol capture with the name of the function.

What to turn in:
Submit your lisp file to the DLWeb website for the course. All assignments are due BEFORE CLASS TIME on the due date.

Hints and Notes:
- Your \texttt{with-list-iterator} and \texttt{doiter} macros should use destructuring in their parameter lists.
- Your macros should protect against multiple evaluation of the input, and should avoid unintentional variable capture by using gensym.
- Use \texttt{labels} or \texttt{flet} to define local functions in question #2.
- The definition of an iterator structure is
  \texttt{(defstruct iterator has-next next)}.
• The code for list-iterator is
  (defun list-iterator (lst)
    (let ((iter (make-iterator)))
      (setf (iterator-has-next iter)
        #'(lambda ()
            (not (null lst))))
      (setf (iterator-next iter)
        #'(lambda ()
            (pop lst)))
      iter))