1 What Would Scheme Display

1. What will Scheme output? If it outputs a list, draw a box-and-pointer diagram. If a part contains multiple expressions, only give the output of the final expression. Assume each expression is evaluated sequentially.

(a) scm> (cons 3 4)

(b) scm> (cons 3 (cons (cons 4 5) nil))

(c) scm> (3 . 4)

(d) scm> '(3 . 4)

(e) scm> (list 3 4 5)

(f) scm> (list 3 '(4 5) 6)
(g) \( \texttt{scm> (define a (list 5 (6) (7)))} \)
\( \texttt{scm> a} \)

(h) \( \texttt{scm> (set-car! a 2)} \)
\( \texttt{scm> (define b (list 3 a))} \)
\( \texttt{scm> b} \)

(i) \( \texttt{scm> (set-cdr! (cdr (cdr a)) 8)} \)
\( \texttt{scm> b} \)

(j) \( \texttt{scm> (define c 2)} \)

(k) \( \texttt{scm> (eval 'c)} \)

(l) \( \texttt{scm> '(cons 1 2)} \)

(m) \( \texttt{scm> (eval '(cons 1 2))} \)
2. Define well-formed, which determines whether \texttt{lst} is a well-formed list or not. Assume that \texttt{lst} only contains numbers and no nested lists.

\begin{verbatim}
; Doctests
scm> (well-formed '())
#t
scm> (well-formed '(1 2 3))
#t
; List doesn't end in nil
scm> (well-formed (cons 1 2))
#f

(define (well-formed lst)

)
\end{verbatim}
3. Define `is-prefix`, which takes in a list `p` and a list `lst` and determines if `p` is a prefix of `lst`. That is, it determines if `lst` starts with all the elements in `p`.

   ; Doctests:
   scm> (is-prefix '() '())
   #t
   scm> (is-prefix '() '(1 2))
   #t
   scm> (is-prefix '(1) '(1 2))
   #t
   scm> (is-prefix '(2) '(1 2))
   #f
   ; Note here `p` is longer than `lst`
   scm> (is-prefix '(1 2) '(1))
   #f

   (define (is-prefix p lst) )
4. Define \texttt{waldo} which takes in a list. If that list contains the symbol \texttt{waldo}, it returns the index where \texttt{waldo} first appears. Otherwise, it returns \texttt{#f}.

\begin{verbatim}
scm> (waldo '(1 4 waldo))
2
scm> (waldo '())
#f
scm> (waldo '(1 4 9))
#f

(define (waldo lst)
  )
\end{verbatim}
5. Implement `double-link`, which takes in a list and replaces the second in each pair of two consecutive items with the first using mutation and returns the mutated list. The first of each pair of consecutive items is unchanged.

```scheme
scm> (define a '(1 2 3 4))
(a
scm> (double-link a)
(1 1 3 3)
scm> a
(1 1 3 3)
scm> (double-link '(c s 6 1 a))
(c c 6 6 a)

(define (double-link lst)
  (if ______________________________

  ___________________________

  (begin _________________________

  ___________________________

  ___________________________

  lst)))
```