NINTH QUIZ

You have 15 minutes from the start of class to complete this quiz. Read the questions with care; work with deliberate speed. Don't give us more than we ask for. The usual instructions apply. Good luck!

Problem 1 (18 points)

Suppose you have a list of book structures, which are defined as usual: `(define-struct book (title author price sold instock))`.

(a) (2 points) Fill in the blanks to complete the definition of this function.

```
;; book-position-by-title: list-of-book string -> number
;; The string is a book's title.  If a book with that title is in
;; the list, return its (zero-based) position number (i.e., the
;; first book is at position 0).  If not, return the length of the list.
(define book-position-by-title
  (lambda (LOB t)
    (cond
      ((empty? LOB) _______________)

      ((string=? t (book-title (first LOB))) _______________)

      (else (_______________(_______________(rest LOB) t))))))
```

(b) (4 points) Now let's generalize this function to check for a match on any field of a book. Complete the definition below; of course you'll need more than one symbol per line.

```
;; book-position: list-of-book (book->X) X -> number
;; The second argument is a field selector for books (book-title,
;; book-author, ...); the third argument is a possible value
;; for the selected field.  Return the (zero-based) position
;; number of the first book in the list for which the selected
;; field's value matches the third argument (or the length of
;; the list if none matches).
(define book-position
  (lambda (LOB field-selector value)
    (cond
      ((empty?

      ((equal?

      (else
```

(c) (4 points) Now suppose we have a vector of books instead of a list. Complete this version of `book-position-in-vector` (whose purpose is the same as above).

```
;; book-position-in-vector:  vector-of-book (book->X) X -> number
(define book-position
  (lambda (VOB field-selector value)
    (local ((define book-position-aux
              (lambda (VOB field-selector value pos)
                (cond
                  ((>= pos (____________ VOB)) ____________)
                  ((equal? ____________ (field-selector (____________ VOB ____________)))
                           ____________)
                  (else (book-position-aux VOB field-selector value (add1 ____________)))))))
      (book-position-aux VOB field-selector value _______________))))
```

(d) (6 points) For each of the four functions defined on the previous page, mark its name below as TR (tail-recursive), RO (recursive (only) and not tail-recursive), or N (not recursive at all); consider each function independently, not including any internal definitions. For any function that is RO, list which section(s) of the code make it non-tail-recursive and (in just a couple of words) why.

`book-position-by-title`

`book-position`

`book-position-in-vector`

`book-position-aux`

(e) (2 points) The `book-position-aux` function uses the accumulator approach; the list-based versions don't. If we could rewrite the list-based versions using the accumulator approach, explain how (you don't have to write the code; just give one English sentence describing the main feature you'd have to add to the code). If we can't use the accumulator approach with the list-based versions, say why not.

Problem 2 (4 points)

Deleting an item from a vector is tricky, because you need to fill in the space where the deleted item was. Another approach is just to change that item to some value (say, `empty`) that you can check later to see that nothing's in that space any more. Below is a function that takes this approach; it uses the predefined function `vector-set!`, which changes in place the specified item in the vector, given its position number.

```
;; set-book-to-empty!: vector-of-books number -> nothing
;; Use vector-set! to change (in place) the book at the
;; specified position to empty.
(define set-book-to-empty!
  (lambda (VOB pos)
    (vector-set! VOB pos empty)))
```

Now, suppose you have a vector of books called `Library` that contains a book whose title is `"HtDP"`. Complete the following expression, using the functions defined above where applicable, to set that book to `empty` in the vector.

```
(set-book-to-empty! Library
```

Problem 3 (3 points)

It could make sense to use "lossy" compression (the kind that throws away some information) on a photograph or audio recording, but we would only use "lossless" compression on text, code, or spreadsheets.

(a) What's a possible advantage of lossy compression that makes up for throwing information away?

(b) Why doesn't that advantage apply to text, code, or spreadsheets?