Informatics 41 • Fall 2009 • David G. Kay • UC Irvine • Your name ____________________

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 (14 points)

Suppose you have a list of restaurant structures, which are defined as usual:

`(define-struct rrant (name cuisine phone dish price)).`

(a) (2 points) Fill in the blanks to complete the definition of this function. (Hint: Consider `add1` or `sub1`.)

```
;; rrant-location-by-name: list-of-rrant string -> number
;; The string is a restaurant's name.  If a restaurant with that name is in
;; the list, return its (zero-based) position number (i.e., the
;; first restaurant is at position 0).  If not, return the length of the list.
(define rrant-location-by-name
  (lambda (RL n)
    (cond
      ((empty? RL) ______________)

      ((string=? n (rrant-name (first RL))) ______________)

      (else (______________(______________(rest RL) n))))))
```

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

```
;; rrant-location: list-of-rrant (rrant->X) X -> number
;; The second argument is a field selector for restaurants (rrant-name,
;; rrant-cuisine, ...); the third argument is a possible value
;; for the selected field.  Return the (zero-based) position
;; number of the first restaurant in the list for which the selected
;; field's value matches the third argument (or the length of
;; the list if none matches).
(define rrant-location
  (lambda (RL field-selector value)
    (cond
      ((empty?

      ((equal?

      (else
```

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

```
;; rrant-location-in-vector:  vector-of-rrant (rrant->X) X -> number
(define rrant-location-in-vector
  (lambda (RV field-selector value)
    (local ((define rrant-location-aux
              (lambda (RV field-selector value pos)
                (cond
                  ((>= pos (___________ RV)) ___________)
                  ((equal? ___________ (field-selector (___________ RV ___________)))
                        ___________)
                  (else (rrant-location-aux RV field-selector value (add1 ___________)))))))
      (rrant-location-aux RV field-selector value ______________))))
```

(d) (4 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.

`rrant-location-by-name`

`rrant-location`

`rrant-location-in-vector`

`rrant-location-aux`

Problem 2 (4 points)

For each of the algorithms or operations described below, check the box corresponding most closely to its complexity (i.e., its O-notation) in the average case.

(a) `(filter p? (map f L))` where `p?` and `f` are functions and `L` is a list of *n* items:

❑ Constant—O(1) ❑ Logarithmic—O(log *n*) ❑ Linear—O(*n*) ❑ Quadratic—O(n^2)

(b) `(vector-ref V n)` where `V` is a vector and `n` is a number between 0 and `(vector-length V)`:

❑ Constant—O(1) ❑ Logarithmic—O(log *n*) ❑ Linear—O(*n*) ❑ Quadratic—O(n^2)

(c) Search for an element in a (balanced) binary search tree containing *n* items:

❑ Constant—O(1) ❑ Logarithmic—O(log *n*) ❑ Linear—O(*n*) ❑ Quadratic—O(n^2)

(d) In a (balanced) binary search tree of *n* restaurants, ordered by the restaurant's name, collecting an alphabetical list of all the restaurants in the tree:

❑ Constant—O(1) ❑ Logarithmic—O(log *n*) ❑ Linear—O(*n*) ❑ Quadratic—O(n^2)

Problem 3 (2 points)

(a) If you need to represent 25 different values (e.g., 25 different colors, or the numbers 0 to 24), can you do it in 4 bits? Why or why not?

(b) Only one of the following could possibly be the ASCII representation of the word "TURKEY". Which one is it? (You do not need to know the actual ASCII codes to answer this correctly.)

A. `0101 0100    0101 0100    0101 0100    0101 0100    0101 0100    0101 0100`

B. `0101    0100    1011    1101    1110    0001`

C. `000 101    000 100    000 011    000 101    000 110    000 001`

D. `0101 0100    0101 0101    0101 0010    0100 1011    0100 0101    0101 1001`

E. `0101 0100    0101 0111    0101 0010    0100 1011`