Informatics 41 • Fall 2004 • David G. Kay • UC Irvine   Your name ______________________________

FIFTH QUIZ

Your student ID ______________________________

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 (4 points)

A binary tree is either empty or `(make-node value left right)`, where value is a number and left and right are binary trees. Complete the definition of `sum-values-in-tree` below. All the parentheses are in the correct places and each blank should be filled by exactly one symbol, function name, or constant.

```
;; sum-values-in-tree: binary-tree -> number
;; Return the sum of all the value fields in the binary tree
(define sum-values-in-tree
     (lambda (BT)
          (cond
              ((_______________ BT) 0)

              (else (_______________

                       (_______________ (_______________ BT))

                       (_______________ BT)

                       (_______________ (_______________ BT)))))))
```

Problem 2 (21 points)

You have just been elected President. You have always hated beets, and your first official act is to decree that no restaurant may serve any dish containing beets. [This is just the same task as locating customers of Amazon.com who have purchased books with "bomb" in the title, but we're familiar with restaurants so we'll stick with that context.]

Remembering fondly your undergraduate study of Scheme, you decide to implement this decree yourself (and, thinking of your successors in office, you decide to parameterize the ingredient to remove). Thus, you define the function `remove-hated-ingredient` (but before you start writing code, read all the following advice).

- We are using `new-rrant` structures as in last week's homework:
  ```
  (define-struct new-rrant (name cuisine phone menu))
  ```
  where name, cuisine, and phone are strings and menu is a list of dish structures:
  ```
  (define-struct dish (name price))
  ```
  where name is a string and price is a number.

- In your answer, you will want to use the function `string-is-in?`; you do not have to define it yourself.
  ```
  ;; string-is-in?: string string -> boolean
  ;; Return true if the first argument occurs anywhere in the second.
  ;; Examples returning true:  (string-is-in? "do" "do re mi")
  ;;      (string-is-in? "re" "do re mi")  (string-is-in? "e m" "do re mi")
  ;; Examples returning false:  (string-is-in? "don't" "do re mi")
  ;;      (string-is-in? "remi" "do re mi")  (string-is-in "do re me fa sol" "do re mi")
  ```

- It will help you to define these two helper functions. For full credit, you *do* have to define these below (but you may use either of them in `remove-hated-ingredient` even if you don't finish defining them).

  ```
  ;; remove-ingredient-from-menu:  string  (listof dish)  ->  (listof dish)
  ;; Return a list of those dishes from the input whose name does not include the string.

  ;; remove-ingredient-from-rrant:  string  new-rrant  ->  new-rrant
  ;; Return a new-rrant like the input new-rrant, but with its menu modified by
  ;; remove-ingredient-from-menu
  ```

- We do not expect you to use `map` or `filter` in your code on this quiz, but you may use them if you're confident enough to let your score depend on it.

```
;; remove-hated-ingredient:  string (listof new-rrant) -> (listof new-rrant)
;; Return the input list of restaurants, with each restaurant modified by removing from
;; its menu every dish that mentions the input string.
```