Hannah Zhang
Table Insert/Delete

```
(define (make-table)
  (cons '* '()))
(define (empty-table? t) (null? (cdr t)))

; check if there is already a key with assoc, if not insert at
front
; order of n, must check if the value is in table first
(define (insert! key value table)
  (let ((record (assoc1 key (cdr table))))
    (if record
        (set-cdr! record value)
        (set-cdr! table
                  ; use cdr bc car table is a dummy node
                  (cons (cons key value) (cdr table)))))
  'ok)
; check if the key matches, then set t to cddr t
; delete is order of n^2 because must check assoc and equal
(define (delete! k t)
  (let ((record (assoc1 k (cdr t))))
    ; only need assoc to check if the key exists, else use t
    (cond ((not record) "key not found")
          ((equal? k (caadr t)) (set-cdr! t (cddr t)))
          (else (delete! k (cdr t)))))
  'yes)
; returns the value associated with key
(define (lookup k t)
  (let ((record (assoc1 k (cdr t))))
    (cond (record (cdr record))
          (else #f))))
; checks if key is equal to key of an element, returns that
whole element
(define (assoc1 key records)
  (cond ((null? records) #f)
        ((equal? key (caar records)) (car records))
        (else (assoc1 key (cdr records)))))
; returns key
```

```
(define (rlookup k t)
  (let ((record (rassoc k (cdr t))))
    (cond (record (car record))
          (else #f))))
; checks if value is equal to value of first element
(define (rassoc value records)
  (cond ((null? records) #f)
        ((equal? value (cdar records)) (car records))
        (else (rassoc value (cdr records)))))
; r delete would check if they are equal using values instead of
keys


(define t (make-table))
(insert! 'a 1 t)
(insert! 'b 2 t)
(insert! 'c 3 t)
(lookup 'b t)
(delete! 'b t)
(lookup 'b t)
```