```
;; Data definitions:

(define-struct node (key val l r))
;; BST (Binary Search Tree) is one of:
;;  - false
;;  - (make-node Integer String BST BST)
;; interp. false means no BST, or empty BST
;;         key is the node key
;;         val is the node val
;;         l and r are left and right subtrees
;; INVARIANT: for a given node:
;;     key is > all keys in its l(eft)  child
;;     key is < all keys in its r(ight) child
;;     the same key never appears twice in the tree

(define BST0 false)
(define BST1 (make-node 1 "abc" false false))
(define BST4 (make-node 4 "dcj" false (make-node 7 "ruf" false false)))
(define BST3 (make-node 3 "ilk" BST1 BST4))
(define BST42
  (make-node 42 "ily"
             (make-node 27 "wit" (make-node 14 "olp" false false) false)
             false))
(define BST10 (make-node 10 "why" BST3 BST42))

#;
(define (fn-for-bst bst)
  (cond [(false? bst) (...)]
        [else
         (... (node-key bst)    ;Integer
              (node-val bst)    ;String
              (fn-for-bst (node-l bst))
              (fn-for-bst (node-r bst)))]))

;; BST Integer -> String or false
;; search bst for key k and produce its val, false if k not found
(check-expect (key-search BST0 1) false)
(check-expect (key-search BST10 60) false)
(check-expect (key-search BST10 14) "olp")

;(define (key-search bst k) "") ;stub

;; Template from BST with additional atomic parameter
(define (key-search bst k)
  (cond [(false? bst) false]
        [else
         (cond [(= k (node-key bst)) (node-val bst)]
               [(< k (node-key bst)) (key-search (node-l bst) k)]
               [else (key-search (node-r bst) k)])]))


;; BST Integer -> String or false
;; search bst for val v and produce its corresponding key, false if v not found
(check-expect (val-search BST0 "olp") false)
(check-expect (val-search BST10 "xyz") false)
(check-expect (val-search BST10 "olp") 14)

;(define (val-search bst v) "") ;stub

;; Template from BST with additional atomic parameter
(define (val-search bst v)
  (cond [(false? bst) false]
        [else
         (cond [(string=? v (node-val bst)) (node-key bst)]
               [(not (false? (val-search (node-l bst) v))) (val-search (node-l bst) v)]
               [(not (false? (val-search (node-r bst) v))) (val-search (node-r bst) v)]
               [else false])]))
```