



[Course](#) > [6b: Mutual Reference](#) > [Quiz](#) > Multiple Choice Quiz

Multiple Choice Quiz

Below is a quiz on the material you've learned this week. Unlike the lecture questions, you will only have one attempt at each question, so think carefully before submitting your answers.

For most questions, we recommend doing your work in DrRacket before submitting your answers.

Question 1

0/1 point (graded)

Consider the following data definition:

```
(define-struct node (key val l r))

;; BST 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(left) child
;;   key is < all keys in its r(right) child
;;   the same key never appears twice in the tree
```

Which of the following data examples are valid BSTs?



```
(define BSTA (make-node 6 "a" false false))
```



```
(define BSTB (make-node 6 "a"
  (make-node 4 "t"
    (make-node 3 "h" false false)
    (make-node 7 "j" false false))
  (make-node 8 "v" false false)))
```



```
(define BSTC (make-node 6 "a"
  (make-node 4 "t"
    (make-node 3 "h" false false)
    (make-node 5 "j" false false))
  (make-node 8 "v" false false)))
```



```
(define BSTD false)
```



```
(define BSTE empty)
```

Explanation

BSTA only has one node, so the invariant holds.

In BSTB, `(make-node 7 "j" false false)` violates the invariant.

In BSTC, the invariant holds.

`false` is an examples of a BST, but `empty` isn't.

Submit

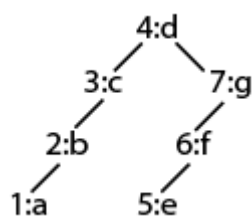
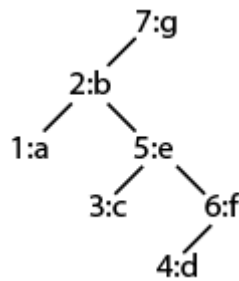
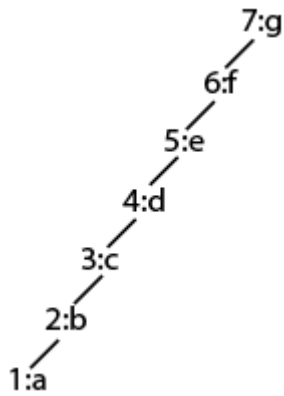
You have used 1 of 1 attempt

i Answers are displayed within the problem

Question 2

1/1 point (graded)

Which of the following images represent valid BSTs, based on the data definition in question 1?



Submit

You have used 1 of 1 attempt

✓ Correct (1/1 point)

Question 3

1/1 point (graded)

Of the above trees, which one should you use to organize your data if you want to improve the average time taken to find a specific key? (Note, for trees this small there will be a negligible difference in speed. Imagine trees much larger, but with the same structure as those shown above).

☐ the first tree

☐ the second tree

☐ the third tree

☒ the fourth tree



Submit

You have used 1 of 1 attempt

✓ Correct (1/1 point)

Question 4

1/1 point (graded)

Suppose you want to design a function called `tree-to-list`, which present the keys in the tree in descending numerical order.

Someone has already completed the signature, purpose, stub and check-expects for you:

```
;; BST -> ListOfInteger
;; present the keys in the binary search tree in descending order as a list
(check-expect (tree-to-list false) empty)
(check-expect (tree-to-list (make-node 4 "e"
                                     (make-node 1 "o" false false)
                                     (make-node 7 "p" false false)))
              (list 7 4 1))
(check-expect (tree-to-list (make-node 6 "a"
                                     (make-node 4 "b"
                                     (make-node 1 "v" false false)
                                     (make-node 5 "s" false false))
                                     (make-node 9 "q"
                                     false
                                     (make-node 10 "x" false
                                     false))))
              (list 10 9 6 5 4 1))
(define (tree-to-list bt) empty) ;stub
```

What is the correct function body for tree-to-list?



```
(define (tree-to-list bt)
  (cond [(false? bt) empty]
        [else
         (cons (tree-to-list (node-l bt))
                 (node-key bt)
                 (tree-to-list (node-r bt))))]))
```



```
(define (tree-to-list bt)
  (cond [(false? bt) empty]
        [else
         (append (tree-to-list (node-l bt))
                  (list (node-key bt))
                  (tree-to-list (node-r bt))))]))
```



```
(define (tree-to-list bt)
  (cond [(false? bt) empty]
        [else
         (append (tree-to-list (node-r bt))
                  (list (node-key bt))
                  (tree-to-list (node-l bt))))]))
```



```
(define (tree-to-list bt)
  (cond [(false? bt) empty]
        [else
         (append (tree-to-list (node-r bt))
                  (node-key bt)
                  (tree-to-list (node-l bt))))]))
```



You have used 1 of 1 attempt

✓ Correct (1/1 point)

Consider the following data definition:

```
(define-struct junction (left straight right))

;; Maze is one of:
;; - false
;; - "finish"
;; - (make-junction Maze Maze Maze)
;; a maze where at each junction you can either go straight, left or
right.
;; false means dead end, "finish" means you've reached the end of the maze
(define M0 false)      ; a maze that is a dead end
(define M1 "finish")   ; a maze where you are already at the finish
(define M2 (make-junction
  (make-junction false
    (make-junction false
      false
      (make-junction false
        false
        false))
    (make-junction false
      (make-junction
        (make-junction false
          false
          false)
        (make-junction false
          "finish"
          false)
        false))
    false)
  false))
  false)) ; a maze
```


Question 5

1/1 point (graded)

What types of reference are present in the type Maze?

☐ none☐ Reference☒ Self-Reference☐ Mutual Reference

You have used 1 of 1 attempt

 Correct (1/1 point)

Question 6

1/1 point (graded)

What is the correct template for Maze?



```
(define (fn-for-maze m)
  (... (junction-left m)
        (junction-straight m)
        (junction-right m)))
```



```
(define (fn-for-maze m)
  (cond [(false? m) (...)]
        [(and (string? m) (string=? "finish" m)) (...)]
        [else
         (... (junction-left m)
               (junction-straight m)
               (junction-right m))]))
```



```
(define (fn-for-maze m)
  (... (fn-for-maze (junction-left m))
        (fn-for-maze (junction-straight m))
        (fn-for-maze (junction-right m))))
```



```
(define (fn-for-maze m)
  (cond [(false? m) (...)]
        [(and (string? m) (string=? "finish" m)) (...)]
        [else
         (... (fn-for-maze (junction-left m))
               (fn-for-maze (junction-straight m))
               (fn-for-maze (junction-right m)))]))
```



You have used 1 of 1 attempt



Correct (1/1 point)

Question 7

1/1 point (graded)

Consider the following templates:

```
(define (fn-for-title t)
  (cond [(string=? t "President") (...)]
        [(string=? t "Vice President") (...)]
        [(string=? t "Director") (...)]
        [(string=? t "Supervisor") (...)]
        [(string=? t "Employee") (...)]))

(define (fn-for-position p)
  (... (position-name p)
        (fn-for-title (position-title p))
        (fn-for-lop (position-subst p))))

(define (fn-for-lop lop)
  (cond [(empty? lop) (...)]
        [else
         (... (fn-for-position (first p))
               (fn-for-lop (rest p)))]))
```

In a terrible coding accident, we have lost the structure definitions, type comments and interpretations for these data types, and we need to reconstruct them from the templates.

What type of data is Title?

☐ Simple Atomic☒ Enumeration☐ Itemization☐ Compound☐ Arbitrary Sized

Submit

You have used 1 of 1 attempt

✓ Correct (1/1 point)

Question 8

1/1 point (graded)

What type of data is `Position` (along with being part of a mutual reference cycle)?

☐ Simple Atomic

☐ Enumeration

☐ Itemization

☒ Compound

☐ Arbitrary-Sized



Submit

You have used 1 of 1 attempt

✓ Correct (1/1 point)

Question 9

1/1 point (graded)

What type of data is `ListOfPosition` (along with being part of a mutual reference cycle)?

☐ Simple Atomic

☐ Enumeration

☐ Itemization

☐ Compound

☒ Arbitrary-Sized



Submit

You have used 1 of 1 attempt

✓ Correct (1/1 point)

Question 10

1/1 point (graded)

What is the correct structure definition and Type Comment for Position?



```
(define-struct position (name title subs))  
;; Position is (make-position String String Position)
```



```
(define-struct position (name title subs))  
;; Position is (make-position String Title Position)
```



```
(define-struct position (name title subs))  
;; Position is (make-position String Title ListOfPosition)
```



```
(define-struct position (name title subs))  
;; Position is (make-position Name Title ListOfPosition)
```



Submit

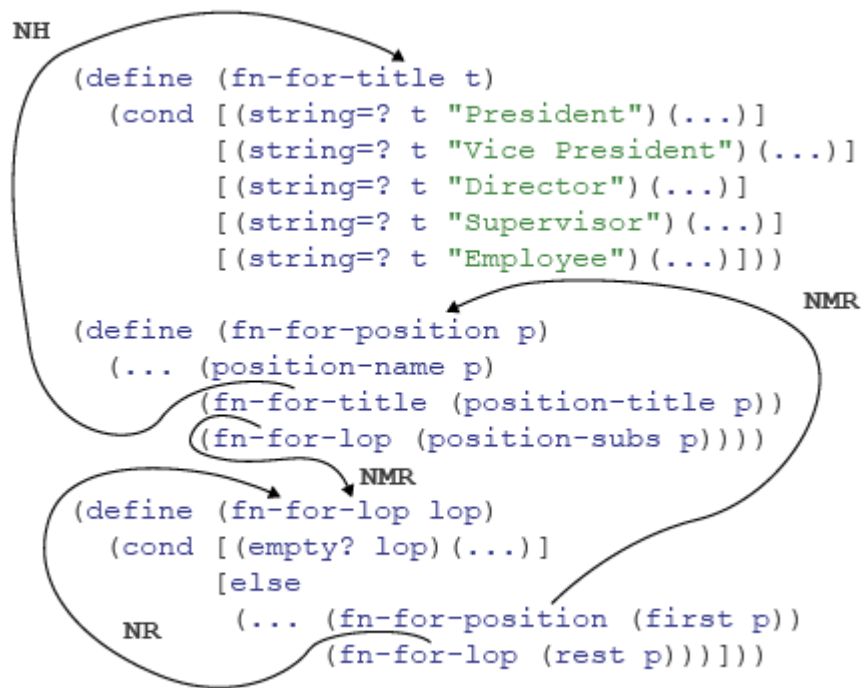
You have used 1 of 1 attempt

✓ Correct (1/1 point)

Question 11

1/1 point (graded)

Correctly label each arrow on the template with one of R, SR, MR, NH, NR, or NMR.



	R	SR	MR	NH	NR	NMR	
--	---	----	----	----	----	-----	--



Submit

You have used 1 of 1 attempt

✓ Correct (1/1 point)

Question 12

1/1 point (graded)

We want to write a function which consumes a name and a position, and produces the position with that name if it exists in the tree under the given position, otherwise false.

Select the correct signature(s), purpose(s) and stub(s)?



```
;; String Position -> Position
;; if a position with the given name exists within the tree, produce that
position, otherwise false
(define (find--position n t p) false) ;stub
```



```
;; String ListOfPosition -> Boolean
;; if a position with the given name exists within the list, produce that
position, otherwise false
(define (find--lop n lop) false) ;stub
```



```
;; String Position -> Position or false
;; if a position with the given name exists within the tree, produce that
position, otherwise false
(define (find--position n p) false) ;stub

;; String ListOfPosition -> Position or false
;; if a position with the given name exists within the list, produce that
position, otherwise false
(define (find--lop n lop) false) ;stub
```



```
;; String Position -> Position or false
;; String ListOfPosition -> Position or false
;; if a position with the given name exists within the tree, produce that
position, otherwise false
(define (find--position n p) false) ;stubs
(define (find--lop n lop) false)
```



Submit

You have used 1 of 1 attempt

✓ Correct (1/1 point)

Question 13

1/1 point (graded)

Which of the following is the correct function definition for `find--position`? Pay close attention to the differences between the options, as some are subtle.



```
(define (find--position n p)
  (if (string=? n (position-name p))
      p
      false))
```



```
(define (find--position n p)
  (if (string=? n (position-name p))
      p
      (find--lop n (position-subst p))))
```



```
(define (find--position n p)
  (if (string=? n p)
      p
      (find--lop (position-subst p))))
```



```
(define (find--position n p)
  (if (string=? n (position-name p))
      p
      (find--lop (position-subst p))))
```



Submit

You have used 1 of 1 attempt



Correct (1/1 point)

Question 14

1/1 point (graded)

Which of the following is the correct function definition for find--lop?



```
(define (find--lop n lop)
  (cond [(empty? lop) false]
        [else
         (if (not (false? (find--position n (first lop))))
             (find--position n (first lop))
             (find--lop n (rest lop)))]))
```



```
(define (find--lop n lop)
  (cond [(empty? lop) false]
        [else
         (if (false? (find--position n (first lop)))
             (find--position n (first lop))
             (find--lop n (rest lop)))]))
```



```
(define (find--lop n lop)
  (cond [(empty? lop) false]
        [else
         (or (find--position n (first lop))
             (find--lop n (rest lop)))]))
```



```
(define (find--lop n lop)
  (cond [(empty? lop) false]
        [else
         (find--position n (first lop))
         (find--lop n (rest lop))])
```



Submit

You have used 1 of 1 attempt

✓ Correct (1/1 point)

