```
(require 2htdp/image)
(require spd/tags)

;; In this problem you will need to remember the following DDs
;; for an image organizer.


;; =================
;; Data definitions:

(@htdd Dir ListOfDir ListOfImage)
(define-struct dir (name sub-dirs images))
;; Dir is (make-dir String ListOfDir ListOfImage)                    A
;; interp. a directory in the organizer, with a name, a list
;;         of sub-dirs and a list of images.


;; ListOfDir is one of:
;;  - empty
;;  - (cons Dir ListOfDir)
;; interp. A list of directories, this represents the sub-directories of
;;         a directory.


;; ListOfImage is one of:
;;  - empty          B
;;  - (cons Image ListOfImage)
;; interp. a list of images, this represents the sub-images of a directory.
;; NOTE: Image is a primitive type, but ListOfImage is not.

(define I1 (square 10 "solid" "red"))
(define I2 (square 12 "solid" "green"))
(define I3 (rectangle 13 14 "solid" "blue"))
(define D4 (make-dir "D4" empty (list I1 I2)))
(define D5 (make-dir "D5" empty (list I3)))    C
(define D6 (make-dir "D6" (list D4 D5) empty))



;; =================
;; Functions:

(@problem 1)
;; Design an abstract fold function for Dir called fold-dir.


(@htdf fold-dir)
(@signature (String Y Z -> X) (X Y -> Y) (Image Z -> Z) Y Z Dir -> X)
;; the abstract fold function for Dir
(check-expect (fold-dir make-dir cons cons empty empty D6) D6)
(check-expect  (local [(define (c1 n rlod rloi) (+ 1 rlod rloi))
                       (define (c2 rdir rlod)   (+ rdir rlod))
                       (define (c3 img rloi)    (+ 1 rloi))]
                 (fold-dir c1 c2 c3 0 0 D6))
               6)

(@template-origin Dir (listof Dir) (listof Image) encapsulated)
                    D
(define (fold-dir c1 c2 c3 b1 b2 d)
  (local [(define (fn-for-dir d)          ; Dir -> X
            (c1 (dir-name d)
                (fn-for-lod (dir-sub-dirs d))
                (fn-for-loi (dir-images d))))
```

```
64
65            (define (fn-for-lod lod)       ; (listof Dir) -> Y
66              (cond [(empty? lod) b1]
67                    [else                  E
68                     (c2 (fn-for-dir (first lod))
69                         (fn-for-lod (rest lod)))]))
70
71            (define (fn-for-loi loi)       ; (listof Image) -> Z
72              (cond [(empty? loi) b2]
73                    [else
74                     (c3 (first loi)
75                         (fn-for-loi (rest loi)))]))]
76      (fn-for-dir d)))
77
78
79  (@problem 2)
80  ;; Design a function that consumes a Dir and produces the number of
81  ;; images in the directory and its sub-directories.
82  ;; Use the fold-dir abstract function.
83
84
85  (@htdf count-images)
86  (@signature Dir -> Natural)
87  ;; count total number of Images in dir and all its subdirs
88  (check-expect (count-images D4) 2)
89  (check-expect (count-images D6) 3)
90
91  (@template-origin use-abstract-fn)
92
93  (define (count-images d)
94    (local [(define (c1 n rlod rloi) (+ rlod rloi))
95            (define (c2 rdir rlod)   (+ rdir rlod))
96            (define (c3 img rloi)    (+ 1 rloi))]
97      (fold-dir c1 c2 c3 0 0 d)))
98                                     F
99
100 (@problem 3)
101 ;; Design a function that consumes a Dir and a String. The function looks in
102 ;; dir and all its sub-directories for a directory with the given name. If it
103 ;; finds such a directory it should produce true, if not it should produce
104 ;; false. Use the fold-dir abstract function.
105
106
107 (@htdf find)
108 (@signature String Dir -> Boolean)
109 ;; look for a dir named name, if found produce true, otherwise produce false
110 (check-expect (find "D4" D6) true)  G
111 (check-expect (find "D8"  D4) false)
112
113 (@template-origin use-abstract-fn)
114
115 (define (find name dir)                    H
116   (local [(define (c1 n rdirs rimgs)
117             (or (string=? name n)
118                 rdirs))
119           (define (c2 rdir rdirs)
120             (or rdir rdirs))
121           (define (c3 img rimgs)
122             false)]
123     (fold-dir c1 c2 c3 false false dir)))
```