```
(require spd/tags)

;; --- OVERVIEW OF PROBLEMS ---
;; 1      Write your CS ID
;; 2 - 4  Fix a buggy function definition.
;; 5      Draw reference arrows.
;; 6      Complete a function definition using for-each.
;; 7      Design a function to produce average rainfall.
;; 8      Design a function to produce a path in a castle
with door locks.

(@problem 1)
;;
;;
;;
;; PUT YOUR CS ID HERE ON THIS LINE
```

```
;; Problems 2 - 4 all involve the following data
definitions.
(@htdd Movie)
(define-struct movie (title revenue genre rating))
;; Movie is (make-movie String Natural String Number)
;; interp. movie with English title, box office revenue in
millions of USD,
;;          genre, and Rotten Tomatoes percentage rating.
(define M1 (make-movie "Titanic" 659 "Romance" 89))
(define M2 (make-movie "Black Panther" 700 "Action" 97))
(define M3 (make-movie "Tomb Raider"  57 "Action" 20))
(define M4 (make-movie "Sound of Music" 163 "Musical" 86))
(define M5 (make-movie "Avengers Infinity War" 678
"Action" 84))
#;
(define (fn-for-movie m)
  (... (movie-title m)
       (movie-revenue m)
       (movie-genre m)
       (movie-rating m)))
```

```
;; Here is a helper function that is used below.  This
helper function
;; is correct and you must not change it.

(@htdf match-genre?)
(@signature Movie String -> Boolean)
;; produce true if m has the genre g
(check-expect (match-genre? M1 "Action") false)
(check-expect (match-genre? M1 "Romance") true)
(check-expect (match-genre? M4 "Musical") true)

(@template Movie add-param)

(define (match-genre? m g)
  (string=? (movie-genre m) g))

;; Problems 2 - 4 all involve different definitions of the
genre-only function.

(@htdf genre-only)
(@signature (listof Movie) String -> (listof String))
;; produce a list of movie titles from lom with g as the
genre
(check-expect (genre-only empty "Romance") empty)
(check-expect (genre-only (list M1 M2 M3 M4 M5) "Action")
              (list "Black Panther" "Tomb Raider"
"Avengers Infinity War"))
(check-expect (genre-only (list M1 M2 M3 M4 M5) "Musical")
              (list "Sound of Music"))




(@problem 2)
;; Below is a structurally recursive function definition.
;; As written it is incorrect. Fix it so that it works
properly.

(@template (listof Movie) add-param)
```

```
(define (genre-only lom g)

  (cond [(empty? lom) empty]

        [else

         (if (match-genre? (first lom) g)

             (cons (movie-title (first lom))

                   (genre-only (rest lom)))

             (genre-only (rest lom)))]))



(@problem 3)
;; Below is a function definition using built-in abstract
functions.
;; As written it is incorrect. Fix it so that it works
properly.

(@template fn-composition use-abstract-fn add-param)

(define (genre-only lom g)

  (map movie-title (filter match-genre? lom)))




(@problem 4)
;; Below is a tail recursive function definition.
;; As written it is incorrect. Fix it so that it works
properly.

(@template (listof Movie) accumulator add-param)

(define (genre-only lom0 g)
  ;; rsf is (listof String); titles of movies with g as
genre seen so far

  (local [(define (fn-for-lom lom rsf)

            (cond [(empty? lom) empty]

                  [else

                   (if (match-genre? (first lom) g)

                       (fn-for-lom (rest lom)

                                   (cons (movie-title
(first lom)) rsf))

                       (fn-for-lom (rest lom)

                                   rsf))]))]

    (fn-for-lom lom0 empty)))
```

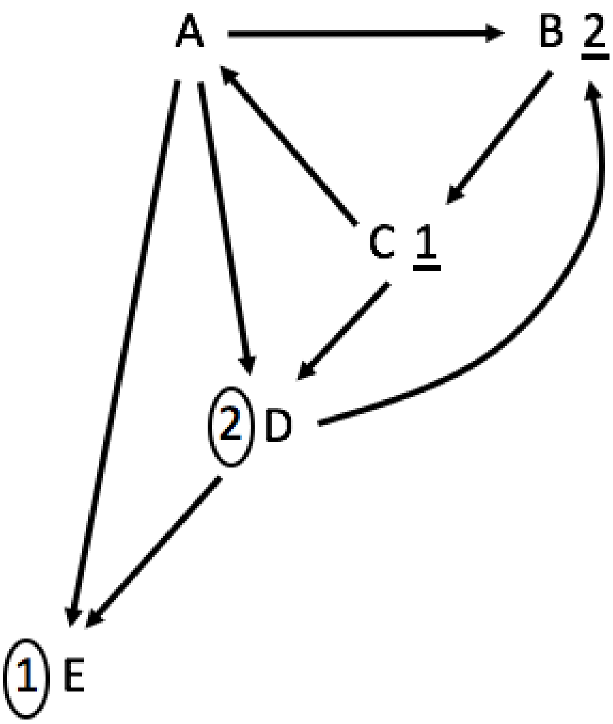
```
(@problem 5)
;;
;; Please draw and label reference arrows on the type
comments below.
;;
;; NOTE:
;;   All arrows must have their "pointy end" pointing to a
type name
;;   that appears right before "is" in a type comment.
;;
;;   All arrows must have their other end clearly starting
from a
;;   reference to a type name.
;;
;;   All arrows should be neatly labeled with R, SR, or MR.
;;
;;
;;   Aisle is (make-aisle Natural ListOfCart)
;;
;;
;;   ListOfCart is one of:
;;
;;     - empty
;;
;;     - (cons Cart ListOfCart)
;;
;;
;;   Cart is (make-cart ListOfItem)
;;
;;
;;   ListOfItem is one of:
;;
;;     - empty
;;
;;     - (cons Item ListOfItem)
```

```
;;
;;
;;    Item is (make-item Image Number Number)
;;
;;
```

```
(@problem 6)
;;
;; In this problem you will complete a function definition
using for-each.
;;
(@htdf skips)
(@signature (listof X) -> (listof X))
;; Skip 1, pick 1, skip 2, pick 1, skip 3, pick 1 ...
(check-expect (skips (list)) (list))
(check-expect (skips (list 1 2 3 4 5 6 7 8 9)) (list 2 5
9))

;; Complete the following function definition below. You
MUST use for-each.
;; You MUST include type and invariant for any
accumulators you use.
(@template for-each accumulator)

(define (skips lox)
  (local [









          ]
    (begin (for-each (lambda (x)






























                       )
                     lox)




           )))

(@problem 7)
;;
;; Design a function called rainfall that consumes a list
of numbers
;; representing daily rainfall amounts as entered by a
user. The list
;; may contain the number -999 indicating the end of the
data of interest.
;; Produce the average of the non-negative values in the
list up to the
```

```
;; first -999 (if it shows up). There may be negative
numbers other than
;; -999 in the list.
;; this page is blank in case you need it for problem 7
```

```
(@problem 8)

;; In this problem you will be working with a simple
representation of a secret
;; castle.  The figure below shows a small secret castle
with 5 rooms named A,
;; B, C, D, and E.  A has exits that lead to rooms B, D
and E.  B has a single
;; exit that leads to C, and so on.  The ovals (long
circles) are locks; the
;; number in the oval is the number of the key required to
open it. The
;; underlined numbers are keys.
;;
;; E has a lock that requires key # 1 to open.  The lock
at room D requires key
;; # 2 to open it. After you get into room B you
automatically pickup key 2.
;; After you get into room C you automatically pickup key
1.
```



```
;;

;; In this castle one legal path from A to the room named
E is A, B, C,
;; D,  E.
;;
;; Note that in general a door might have multiple locks,
and a room might
;; provide multiple keys. A door with a lock might also
provide keys.
;;
;; Here are the data definitions we use.

;; Data definitions:

(@htdd Room)
(define-struct room (name locks keys exits))
;; Room is (make-room String (listof Natural) (listof
Natural) (listof Room))
;; interp. each room has
;;             - a Name
;;             - locks that require a key of the same number
to open
;;             - keys that open locks of the same number
;;             - doors to other rooms
```

```
;;
;; NOTE: The keys can be for any rooms in the castle, they
do not have
;;       to be for one of the rooms in exits.
;;

(define CASTLE
  (shared ([-A- (make-room "A" (list)   (list)   (list -B-
-D- -E-))]
           [-B- (make-room "B" (list)   (list 2) (list
-C-))]
           [-C- (make-room "C" (list)   (list 1) (list -A-
-D-))]
           [-D- (make-room "D" (list 2) (list)   (list -B-
-E-))]
           [-E- (make-room "E" (list 1) (list)   (list))])

    (list A B C D E)))

(define A (list-ref CASTLE 0))
(define B (list-ref CASTLE 1))
(define C (list-ref CASTLE 2))
(define D (list-ref CASTLE 3))
(define E (list-ref CASTLE 4))

;; Design a function that consumes a Room and a String and
tries to find
;; a path from the room to a room with the given name.  A
path
;;   - cannot go into a room more than once
;;   - cannot go into a room unless the required keys are
being held
;; You can only pick up the keys in a room once you are
inside it.  You may
;; carry more than one key at a time.
;;
;; If a path is possible your function should produce the
list of room names
;; traversed in order.
;;    (find-path A "E") should produce (list "A" "B" "C"
"D" "E")
;;    (find-path D "E") should produce false
;;
;; Your design must include signature, purpose,
appropriate tests, @template
;; tag, and working function definition.  Your design must
follow templating
;; and all other applicable design rules.  Any
accumulators must have type and
;; invariant comments.
;;
;; Your function should NOT use tail recursion.  Just use
ordinary structural
;; recursion.  We will mark solutions that use tail
recursion, but the function
;; is harder to write that way so we advise against it.
;;
;; PLEASE TAKE YOUR TIME AND WRITE NEATLY.

;; Use this page and the next one for problem 8.
```

```
;;This page left blank in case you need it for problem 8.
```