

Exercise 4.41.

Write an ordinary Scheme program to solve the multiple dwelling puzzle.

Answer.

An crucial decision must be made to solve the puzzle in ordinary Scheme is how to represent the dwelling assignment in Lisp. Since a person merely dwells on a single floor, while a floor is able to accomodate many people conceptually, it becomes natural to center our discussion around people rather than floors. We choose to denote the allocations of these 5 floors by a sequence of 5 numbers. For example the list (4 5 3 1 2) indicates that Baker lives on the fourth floor, Cooper occupies the top floor, Fletcher resides on the middle of the building, Miller sits on the ground, and Smith sattles on the second floor.

```
(define (baker sequence)
  (car sequence))

(define (cooper sequence)
  (car (cdr sequence)))

(define (fletcher sequence)
  (car (cdr (cdr sequence))))

(define (miller sequence)
  (car (cdr (cdr (cdr sequence)))))

(define (smith sequence)
  (car (cdr (cdr (cdr (cdr sequence))))))
```


A natural strategy to determine the distribution of floors is to generate all the way of arranging them around those 5 individuals, that is, the permutations of a set $S = \{1, 2, 3, 4, 5\}$, filter to select those sequences which are qualified for the designated restrictions. Since elements of S are unique, individuals will automatically be guaranteed to live on different floors.

```
(define (multiple-dwelling)
  (let ((floors (list 1 2 3 4 5)))
    (let ((init-assigns (permutations floors)))
      (let ((accept-assigns (filter legitimate? init-assigns)))
        (for-each (lambda (assignment) (print-assign assignment))
                  accept-assigns)))
    'ok)

(define (legitimate? assignment)
  (and (not (= (baker assignment) 5))
       (not (= (cooper assignment) 1))
       (not (= (fletcher assignment) 5))
       (not (= (fletcher assignment) 1))
       (> (miller assignment) (cooper assignment))
       (not (= (abs (- (smith assignment) (fletcher assignment))) 1))
       (not (= (abs (- (fletcher assignment) (cooper assignment))) 1))))
```

The procedure `print-assign` displays the result in the same way as the `amb` evaluator did:

```
(define (print-assign assignment)
  (display (list (list 'baker (baker assignment))
                 (list 'cooper (cooper assignment))
                 (list 'fletcher (fletcher assignment))
                 (list 'miller (miller assignment))
```

*. Creative Commons  2014, Lawrence X. Amlord (颜世敏, aka 颜序).
Email address: informlarry@gmail.com

```
(list 'smith (smith assignment))))
```

Evaluating the expression (multiple-dwelling) produces the same result to the text

```
(multiple-dwelling)  
((baker 3) (cooper 2) (fletcher 4) (miller 5) (smith 1))  
;Value: ok
```