Exercise 4.40.

In the multiple dwelling problem, how many sets of assignments are there of people to floors, both before and after the requirement that floor assignments be distinct? It is very inefficient to generate all possible assignments of people to floors and then leave it to backtracking to eliminate them. For example, most of the restrictions depend on only one or two of the person-floor variables, and can thus be imposed before floors have been selected for all the people. Write and demonstrate a much more efficient nondeterministic procedure that solves this problem based upon generating only those possibilities that are not already ruled out by previous restrictions. (Hint: This will require a nest of let expressions.)

Answer.

Before the requirement that floor assignments are distinct, there are 120 sets of assignments. (Hint: Based on their dependency, enumerate the possible assignments in the order of Cooper, Fletcher, Smith, Miller and Baker.) Once this requirement is imposed, there remains only 1 legitimate assignment—the one the interaction in the text shows.

If impossible allocations of individuals could be ruled out before assignments of people to floors are generated, the efficiency of computation will leap giantly, for much of the backtracking for elimination can be avoided. The following `multiple-dwelling` procedure is based upon generating only those possibilities that are not already ruled out by previous restrictions and is expected to run much more faster than all its predecessors.

```
(define (multiple-dwelling)
  (let ((cooper (amb 2 3 4))
        (fletcher (amb 2 3 4)))
    (require (not (= (abs (- fletcher cooper)) 1)))
    (let ((miller (amb 3 4 5)))
      (require (> miller cooper))
      (let ((smith (amb 1 2 3 4 5)))
        (require (not (= (abs (- smith fletcher)) 1)))
        (let ((baker (amb 1 2 3 4)))
          (require
           (distinct? (list baker cooper fletcher miller smith)))
          (list (list 'baker baker)
                (list 'cooper cooper)
                (list 'fletcher fletcher)
                (list 'miller miller)
                (list 'smith smith)))))))
```