COMP 210, FALL 2000
Extra Credit Homework (10 points)
Due date: Friday, April 19, 2002

When medical students graduate from medical school, they must apply to hospitals for residency programs. Obviously, the students want to get into the hospital program that best suits their needs, and the hospitals want to get the best students they can.

The problem, then, becomes how to assign students to hospitals in such a way that the people and hospitals are as happy as possible. We call this a "stable" relationship: when no hospital would rather have a different student, and that student (assigned to another hospital) would rather have the first hospital.

Let's look at an example: we'll assume that we have three hospitals and three students. Everybody makes a list of their preferences as such:

Student1 {HospitalA, HospitalC, HospitalB}
Student2 {HospitalA, HospitalC, HospitalB}
Student3 {HospitalA, HospitalB, HospitalC}

HospitalA {Student3, Student2, Student1}
HospitalB {Student1, Student3, Student2}
HospitalC {Student3, Student2, Student1}

An unstable pairing would be:

HospitalA gets Student1
HospitalB gets Student2
HospitalC gets Student3

...because HospitalA could rather have StudentC, and StudentC would also rather have HospitalA.

A stable pairing would be:

HospitalA gets Student3
HospitalB gets Student1
HospitalC gets Student2

Check for yourself that this is stable.

An obvious question to ask from this is, "will there always be a stable set of pairings?" It turns out that the answer is yes, there will always be an answer, as long as each student and each hospital's preference list contains all of the possible choices.

There are a couple of different ways to solve this problem, some more efficient than others. One way, of course, would be to try every possible set of pairings and just find one that is stable, much like our solution to the missionaries and cannibals problem.

However, we can implement this much more efficiently. The classic algorithm to solve this problem starts with the first hospital and gives it its first pick. We then go to the second hospital and assign it its first pick. If the first and second hospitals list the same student as their top pick, then we need to decide which one gets that student. Clearly, whichever of the two hospitals is more highly favored by the student is the one that should get the student.

This, then, is our algorithm:

For each hospital,
assign the first student to the hospital if the student is either: 1) not already assigned to another hospital, or 2) if the student would prefer the current hospital to the one to which he is already assigned. If we moved the student from one hospital to another, we clearly have to go find a new student for the second hospital. If the hospital's first choice is unavailable, we move to the hospital's second choice, etc.

We suggest you use the following data structures to represent hospitals and students:

```
; a student is a data structure
; (make-student name assignment preferences)
; where name is a symbol
;       assignment is a symbol
;           preferences is a list of symbols

; a hospital is a data structure
; (make-hospital name assignee preferences)
; where name is a symbol
;       assignee is a symbol
;           preferences is a list of symbols
```

For the "assignment" and "assignee" fields, we suggest that you make up a special symbol to denote that the hospital or student does not yet have a choice made for them.

In the original example, our data structures would look like:

```
(define list_of_students (list
   (make-student 'Student1    'UNASSIGNED (list 'HospitalA, 'HospitalC, 'HospitalB))
   (make-student 'Student2    'UNASSIGNED (list 'HospitalA, 'HospitalC, 'HospitalB))
   (make-student 'Student3    'UNASSIGNED (list 'HospitalA, 'HospitalB, 'HospitalC))))

(define list_of_hospitals (list
   (make-hospital 'HospitalA 'UNASSIGNED (list 'Student3, 'Student2, 'Student1))
   (make-hospital 'HospitalB 'UNASSIGNED (list 'Student1, 'Student3, 'Student2))
   (make-hospital 'HospitalC 'UNASSIGNED (list 'Student3, 'Student2, 'Student1))))
```

One function we obviously will need is one that tells us the ordinal position in a student or hospital's list of preferences -- so that we can answer the question, for example, "who does 'Student1 prefer?" Also, we will need some way to look up a name in a list of students or hospitals and have that person's record returned.

For this assignment, you will need to use "set" routines extensively -- you may not create new lists of either students, hospitals, or preferences; you must make any changes in place.

Your code must include the following two functions:

```
; stable-picks: list-of-hospital list-of-student -> void
; Purpose: takes a list of hospitals and students and finds a stable
;          pairing
; Effects: updates the lists in place
(define (stable-picks aloh alos) ... )

; print-assignments: list-of-hospital -> void
; Purpose: prints the pairings generated by stable-picks and prints
;          them out in a pairwise fashion as:
;          'HospitalA  gets   'Student1
;          ...etc.
```