MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Spring Semester, 1998

Problem Set 4

- Issued: Tuesday, February 24
- Tutorial preparation for: Week of March 2
- Written solutions due: Friday, March 6 in recitation
- Reading: Read section 2.3 before lecture on February 26. Read section 2.4 before lecture on March 5.
- Reminders: (1) The 6.001 Help Line and the Homework discussion forum are available sources of help on problem sets. See the 6.001 web page for details. (2) Quiz 1 will be held on Wednesday evening, March 11. More information will be distributed in lecture on Tuesday, March 3.

The Freshthing Housing Assignment System

The following fragment of a confidential memo was discovered in the trash on the second floor of Building 3.

> . . . to announce the fruition of the long-range plan, initiated by the Killian administration in the 1950s, to expand MIT's sources of revenue, increase our pool of outstanding students, and augment the diversity of our student body, by opening admissions to sentient beings applying from anywhere in the Universe.
>
> As you know, MIT admissions applications were encoded into the original television transmissions of "I Love Lucy," which resulted in them being widely broadcast throughout the galaxy. We've just received word from our friends at the SETI project that we should expect to matriculate a large number of outstanding freshthings in the school year beginning September 1998.
>
> We will be enrolling a diverse group of freshthings from a variety of extraterrestrial communities. Some of these freshthings can only comfortably survive in a reducing atmosphere of methane. For some of them we need to maintain temperatures between 350K and 370K. Some of them can be comfortable only when submerged in a solution of ethanol. This last case is particularly worrisome, because maintaining an appropriate environment for our ethanol-breathing freshthings violates our published alcohol policy. I am sure we will muddle through. However, we need to act quickly to understand the problem so we can begin construction of appropriate facilities in Random Hall.
>
> We will also have to arrange with Food Services to provide ...

The rest of the memo was missing, but you can see the problem. This assignment is an attempt to take the first steps in sorting "things" out.

1. Representing housing preferences

Each incoming freshthing has preferences for his/her/its living space. For example, the atmosphere should have the correct composition and the temperature range should be correct. There are other kinds of preferences, such as the desirable spectral range of electromagnetic radiation in which the freshthing sees, and the diurnal phase that is appropriate for its work or study.

Some kinds of preferences are more important than others. For example, having a roommate who prefers to be awake when you are asleep is merely annoying, but having to room with someone who breathes an atmosphere that is poisonous to you would be rather more problematic. We will therefore organize the preferences into categories and specify an order of importance to the categories. Every freshthing will pick housing preferences in each category. The categories, in order of importance, and the initial possible choices are:

- **Atmosphere:** methane, oxygen, sulfur, ethanol, etc.
- **Temperature:** 350K-370K, 290K-300K, 260K-270K, etc.
- **Spectrum:** infrared, visible, ultraviolet, etc.
- **Cycle:** day, night, random, etc.

When a new freshthing interacts with the Housing Office, he/she/it indicates preferences for items in this list. A freshthing may list as many preferences as it wishes, but it may not list a preference in a category without listing preferences in all the categories that are more important. For example, it might list *(methane, 260K-270K, infrared, random)*, or *(oxygen, 290K-300K, visible)*, or *(sulfur, 350K-370K, ultraviolet)*. But it may not list *(sulfur, ultraviolet)*, because that skips temperature, which is more important than spectrum.

Our goal is to implement a system that will take all the preferences from the incoming class, group together those students with similar needs, then use the results to figure out which students to group together and how much reconstruction of Random Hall will be required to convert rooms to the appropriate environments.

In order to keep track of the preferences, we will use a data structure called an *association table.* Such a table associates a sequence of keys with a list of values. In our application, the sequence of keys will be the preferences specified, and the list of values will the the list of freshthings who specified exactly those preferences.

We can regard such a table as a tree, as shown in figure 1. Each node of the tree (indicated by black dots in the figure) designates a list of values (indicated by boxes in the figure). The key sequence to access any of the lists of values in the tree are the labels on the edges of the tree as we move from the root to the appropriate node.

For example, in figure 1, if we follow the sequence of keys (*methane*, *260K-270K*), we find that the freshthing named Hexapod4 requires a methane atmosphere and a temperature range of of 260K-270K. If we extend the key sequence to be (*methane*, p*260K-270K*, *infrared*), there is no freshthing who has exactly that set of preferences (as represented by the empty box). But if we

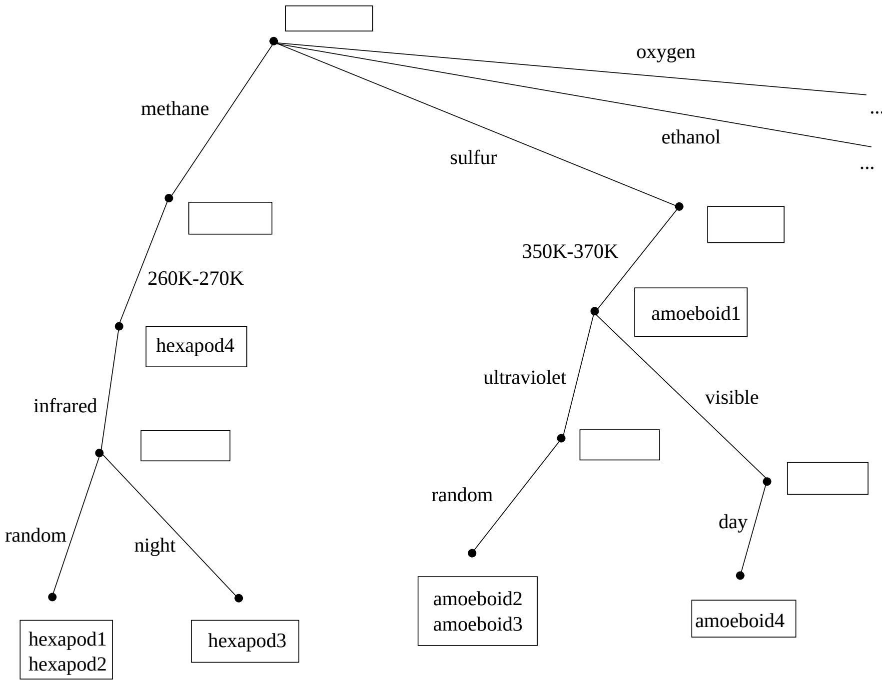

Figure 1: Freshthing housing preferences, viewed as a tree structure

extend the sequence further to be (*methane*, *260K-270K*, *infrared*, *random*), we find that these are the preferences for Hexapod1 and Hexapod2. Similarly, we can examine the branch for freshthings who require a sulfur atmosphere, or oxygen, or ethanol. Note the empty box at the top of the tree, which signifies that there are no freshthings who have expressed no preferences (although expressing no preferences would be an OK thing to do).

We will represent tables using list structure. A table is a list whose first element is the list of values at the node at the top of the table and whose remaining elements are the branches emanating from that node. To make an empty table, we use empty lists for the keys and the branches:

```
(define (make-table values list-of-branches)
  (cons values list-of-branches))

(define table-values car)
(define table-branches cdr)

(define (make-empty-table)
  (make-table '() '()))
```

A branch is a pair consisting of a key (the label on the branch) and a subtable (the node that the branch connects to):

```
(define (make-branch key subtable)
  (cons key subtable))
```

```
(define branch-key car)
(define branch-subtable cdr)
```

The essential building block in our system is a method for adding a new value to a table, associating it with a specified sequence of keys:

```
(define (add-to-table val table keys)
  (if (null? keys)
      ;; keys ran out so insert new value here.
      (make-table (cons val (table-values table))
                  (table-branches table))
      ;; more keys to go, so descend deeper into tree.
      (make-table (table-values table)
                  (insert-in-branches val
                                      (table-branches table)
                                      keys))))
```

The subprocedure `insert-in-branches` checks the list of branches to find the one with the right key (i.e., the first key in the sequence), and adds the value to the structure for that branch:

```
;; Assumes there is at least one key
;; Search the branches looking for a match on the first key.
;; Return the new list of branches with the appropriate subtable augmented.
(define (insert-in-branches val branches keys)
  (cond ((null? branches)
         ;; Make a new list with one new branch.
         (list
          (make-branch (car keys)
                       (add-to-table val
                                     (make-empty-table)
                                     (cdr keys)))))
        ((eq? (car keys) (branch-key (car branches)))
         ;; found the right branch to augment: Add the value to
         ;; the subtable for that branch
         (cons
          (make-branch (branch-key (car branches))
                       (add-to-table val
                                     (branch-subtable (car branches))
                                     (cdr keys)))
          (cdr branches)))
        (else (cons (car branches)
                    (insert-in-branches val (cdr branches) keys)))))
```

Notice the (tricky!) intertwined recursive structure here: `add-to-table` calls `insert-in-branches` to insert the element into the appropriate branch; `insert-in-branches` calls `add-to-table` to insert the element into the subtable for the appropriate branch.

The following procedure begins with a list of freshthings and their preferences (assumed in correct order and not skipping any) and builds up an association table, installing the freshthings one by one:

```
(define (install table things)
  ;; things is a list of freshthings
  ;; each is a list of name and a list of attributes
  (cond ((null? things)
         table)
        (else
         (install (add-to-table (car (car things))
                                table
                                (cadr (car things)))
                  (cdr things)))))
```

For example, our initial galactic entering class of 1998 might be:

```
(define entering-class
  (install (make-empty-table)
           '((hexapod1 (methane 260K-270K infrared random))
             (hexapod2 (methane 260K-270K infrared random))
             (hexapod3 (methane 260K-270K infrared night))
             (hexapod4 (methane 260K-270K))
             (amoeboid1 (sulfur 350K-370K))
             (amoeboid2 (sulfur 350K-370K ultraviolet random))
             (amoeboid3 (sulfur 350K-370K ultraviolet random))
             (amoeboid4 (sulfur 350K-370K visible day))
             (gnork (ethanol 290K-300K))
             (gnorkette (ethanol 290K-300K ultraviolet))
             (snork (ethanol 290k-300k))
             (tork (ethanol 260K-270K))
             (bork (ethanol 260K-270K))
             (fred (oxygen 290K-300K visible))
             (mary (oxygen 290K-300K visible day))
             (mike (oxygen 290K-300K)))))
```

The resulting structure, printed as a list is:

```
(()
 (methane ()
          (260k-270k (hexapod4)
                     (infrared ()
                               (random (hexapod2 hexapod1))
                               (night (hexapod3)))))
 (sulfur ()
         (350k-370k (amoeboid1)
                    (ultraviolet ()
                                 (random (amoeboid3 amoeboid2)))
                    (visible ()
                             (day (amoeboid4)))))
 (ethanol ()
          (290k-300k (snork gnork)
                     (ultraviolet (gnorkette)))
          (260k-270k (bork tork)))
```

```
 (oxygen ()
        (290k-300k (mike)
                   (visible (fred)
                            (day (mary))))))
```

This is in fact the table shown in figure 1, with the branches for the ethanol and oxygen breathers included.

2. Tutorial Exercises

We strongly suggest that you do all of the tutorial exercises before starting work on the computer exercises. This will give you valuable practice thinking in terms of aggregate operations. There's also some important design work that you'll need to do before diving into programming.

Tutorial exercise 1: Do exercises 2.33 and 2.35 of the textbook.

Tutorial exercise 2: Do exercise 2.53 of the textbook.

Tutorial exercise 3: Consider the following sequence of evaluations

```
(define *table* (make-empty-table))

(define *table*
  (add-to-table 'hexapod1 *table* '(methane 260K-270K infrared random)))

(define *table*
  (add-to-table 'hexapod2 *table* '(methane 260K-270K infrared random)))
```

Using box-and pointer notation, trace through the creation of the table, showing the calls to `add-to-table` and `insert-in-branches` at each step of the construction.

Tutorial exercise 4: Alien freshthings, even though they are MIT students, are sometimes not very careful about following directions. (Several faculty members have described them as "spacey.") In particular, the Housing Office is concerned that some of them might not be careful about listing their preferences in decreasing order of importance, not skipping any categories. Describe how present system behaves if some freshthings violate this constraint. What happens, for example, if incoming student cxthlogtl lists its preferences as *(night, sulfur, ultraviolet)*?

Tutorial exercise 5: Design a procedure called `canonicalize-input`, which takes an input a freshthing in some more flexible form and cleans up the result, generating a list that could be used for that freshthing in the `install` procedure. There is no single correct answer here. You will need to define the format of the input and the range of inputs you'll permit. For example, you might have each freshthing supply a list of a name followed by pairs indicating the preferences, such as:[1]

```
(cxthlogtl (cycle night) (atmosphere sulfur) (spectrum ultraviolet))
```

For this exercise, you needn't implement the procedure—just design it. In tutorial, be prepared to say what the input format your program will accept and describe how you will convert this to a valid input for our system. For example, how will you handle an input where that skips a category or lists things out of order? Describe your program as clearly and as elegantly as possible. Remember that programs like these are often elegantly written using `map`, `filter`, `accumulate` aggregate operations. Assume that the overall order of importance of categories is specified in the housing system by the list `*order*`:

```
(define *order* '(atmosphere temperature spectrum cycle))
```

3. Programming assignment: So what can room with what?

Our ultimate goal in this problem set is to help the Housing Office determine what incoming freshthings can be put in the same room, and to determine what dorms will need to be converted to a different environment (note that some of them may already have an appropriate atmosphere for some of these new students!).

Begin by loading the code for problem set 4. This defines the procedures listed above, and the list `*order*` and the initial database `entering-class`. The loaded code also defines `map`, `filter`, and `accumulate` from the book, which you may find useful in doing the exercises below. In grading problem sets, we will be considering the elegance of your code, not only whether or not it "works." So it will be to your advantage to use aggregate operations when they are appropriate. In fact, some of the exercises below will be very cumbersome to write (and debug!) if you do not use the aggregate operators. It would be a good idea to reread pages 112–118 of the book, and to look again at exercise 2.35 before starting in on these programming exercises.

Computer exercise 1: Implement a procedure that, given a table, returns the list of all the freshthings in the table. Note that the result should be a single flat list (not a list with sublists). Test your procedure by finding all the freshthings in `entering-class`.

Computer exercise 2: Implement a procedure that takes as argument a table and a list of conditions and returns the list of all freshthings in the table who could live under those conditions. Assume that if a freshthing does not list a preference in a category, then he/she/it will be comfortable with any choice.

[1] Such a list structure could be generated by checking off boxes using a form on the Galactic Wide Web.

For instance, with our sample entering class, if the list of conditions is *(sulfur, 350K-370K, ultraviolet)*, your procedure should return all freshthings who could live with a sulfur atmosphere, a 350K-370K temperature range, and ultraviolet light—namely, Amoeboid1, Amoeboid2, and Amoeboid3. Note that this result does not include Amoeboid4, since Amoeboid4 specified visible light; but the result does include Amoeboid1, since Amoeboid1 did not specify any preference for spectrum. Assume that the list of conditions starts with an atmosphere preference, lists the preferences in order of importance, and does not skip over any categories. That is, the list could be *(methane)* or *(methane, 260K-270K, infrared, random)*, but not *(260K-270K)* or *(methane, infrared, random)*. Turn in a listing of your procedure together with some tests to demonstrate that it works.

By the way, the Boston Licensing Board has just informed MIT that it will not allow students to live in ethanol-atmosphere housing in the future. How would you use your procedure to find out how many students could be displaced by this ruling?

Computer exercise 3: Generalize your answer to exercise 2 so that the list of conditions can include the symbol `*` to indicate that a category is skipped over. For example, using *(sulfur, *, visible)* should return all freshthings except those who specified an atmosphere other than sulfur or a spectrum other than visible—namely, Amoeboid1 and Amoeboid4. Similarly, *(*, *, ultraviolet)* should return the list of all freshthings who can live in an ultraviolet spectrum, i.e., all freshthings except those who specified a preference for a spectrum other than ultraviolet. (There are 10 such freshthings in our sample entering class.)

Note that the preferences must still list the categories in order and include a `*` for each category that is skipped over. Using your program, find all the freshthings who could live with an ultraviolet spectrum and a night cycle. Find all the freshthings who could live with a day cycle.

In this problem, especially, it will really help to think in terms of aggregate operations. If you don't do this, it is easy to get lost in a maze of list operations. So plan your work carefully.

Computer exercise 4: Implement the `canonicalize-input` procedure you designed for tutorial exercise 5, and turn in the procedure together with some tests to show that it works.

Computer exercise 5: Our system currently requires freshthings to indicate their preferences for categories using our ordering of importance (atmosphere, temperature, spectrum, cycle). But this is too rigid. For instance, my friend Gleep, from the second planet around Proxima Centauri, is comfortable over a reasonably wide range of temperatures and atmospheres, but Gleep really needs to be strongly illuminated with visible light because he is photosynthetic. Wouldn't it be better to allow freshthings to give only the characteristics that actually matter to them, independent of order?

Discuss how you would redesign the system to permit this. The new system will need to accept input from freshthings, enter this into the table, and access the table to find all freshthings who can live under various conditions (as in exercise 3 above).

Write a clear but brief (one or two page) explanation describing which parts of the system need to be changed, and how. This is a very open-ended question and there are lots of ways to go about this. In grading, we will be more interested in the clarity of your explanation than in details of coding. For example, you should describe any new procedures that must be written and any old procedures that must be modified and give precise prescriptions of how they should behave.

If you want to implement some or all of your design, that's fine, but you need not write any code at all for this problem, provided you give a clear explanation of how the new system will work and a clear description of what code needs to be written. "Clear" means that another person, say another student in the class, could read your description and write the actual code without difficulty.

Turn in answers to the following questions along with your answers to the questions in the problem set:

1. About how much time did you spend on this homework assignment? (Reading and preparing the assignment plus computer work.)
2. Which scheme system(s) did you use to do this assignment (for example: 6.001 lab, your own NT machine, your own Win95 machine, your own Linux machine)?
3. We encourage you to work with others on problem sets as long as you acknowledge it (see the 6.001 General Information handout).
   - If you cooperated with other students, LA's, or others, or found portions of your answers for this problem set in references other than the text (such as some of the archives), please indicate your consultants' names and your references. Also, explicitly label all text and code you are submitting which is the same as that being submitted by one of your collaborators.
   - Otherwise, write "I worked alone using only the reference materials," and sign your statement.