

## Laboratory: Association Lists

**Summary:** In today's laboratory, you will experiment with association lists, structures that make it easy to look up information.

### Contents:

- Exercises
  - Exercise 1: Science Chairs, Revisited
  - Exercise 2: Science Chairs' Departments
  - Exercise 3: Cartoon Sidekicks
  - Exercise 4: Finding Sidekicks
  - Exercise 5: Duplicate Keys
  - Exercise 6: Preconditions
  - Exercise 7: Reverse Associations
  - Exercise 8: Using a Specific Database
- For Those with Extra Time
  - Extra 1: Lists as Keys
  - Extra 2: Compound Keys
- Notes
  - Notes on Exercise 3

**Useful procedures:** `assoc`, `liist-ref`, `member`.

## Exercises

### Exercise 1: Science Chairs, Revisited

In the reading on association lists, we claimed that the `lookup-telephone-number` procedure worked correctly, even for the extended table that included not only phone number, but also department.

Verify that claim.

You can find the table and the code in the reading.

### Exercise 2: Science Chairs' Departments

Document and write a procedure, `(lookup-department name directory)`, that someone who knows the name of the chair of a department can use to find that department's name.

### Exercise 3: Cartoon Sidekicks

Define an association list, `sidekicks`, that associates the names of some famous cartoon protagonists (as strings) with the names of their sidekicks (again, as strings).

You may also want to look at the note on this exercise.

Here's a table containing information for your association list:

Protagonist	Sidekick
Peabody	Sherman
Yogi	Booboo
Secret Squirrel	Morocco Mole
Tennessee Tuxedo	Chumley
Quick Draw McGraw	Baba Looey
Dick Dastardly	Muttley
Bullwinkle	Rocky
Bart Simpson	Milhouse Van Houten
Asterix	Obelix
Strong Bad	The Cheat

### Exercise 4: Finding Sidekicks

Use the `assoc` procedure to search the `sidekicks` association list for someone who is on the list and for someone who is not on the list.

### Exercise 5: Duplicate Keys

- Redefine `sidekicks` so that it includes two entries with the same protagonist and different sidekicks -- say Scooby Doo, who has both Shaggy and Scrappy Doo as sidekicks. What do you expect to happen if you try to apply `assoc` to retrieve these entries, using the common key "Scooby Doo"?
- Check your answer experimentally.
- Many people find these results disappointing. To help alleviate this disappointment, define and test a procedure, (`multi-assoc key alist`), similar to `assoc`, except that it returns a list of *all* the lists with the given key.

## Exercise 6: Preconditions

a. What do you think that `assoc` will do if it is given a list in which each element is a pair, rather than a list? For example, can we use `assoc` to search the following list to determine the first name of a faculty member whose last name you know?

```
(define some-faculty
  (list (cons "Walker" "Henry")
        (cons "Stone" "John")
        (cons "Rebelsky" "Sam")
        (cons "Davis" "Janet")
        (cons "Coahran" "Marge")
        (cons "Adelberg" "Arnie")
        (cons "Herman" "Gene")
        (cons "Jepsen" "Chuck")
        (cons "Moore" "Emily or Tom")
        (cons "Wolf" "Royce")
        (cons "Chamberland" "Marc")
        (cons "Shuman" "Karen")
        (cons "French" "Chris")
        (cons "Kornelson" "Keri")
        (cons "Romano" "David")
        (cons "Mosley" "Holly")
        (cons "Kuiper" "Shonda")
        (cons "Jager" "Leah")))
```

b. Confirm or refute your answers by experimentation.

c. Based on your experience, what preconditions should `assoc` have?

## Exercise 7: Reverse Associations

a. What happens if you search the hero/sidekick list by sidekick instead of by protagonist? For example, you might try

```
(assoc "Chumley" sidekicks)
```

b. Define and test a procedure `lookup-by-second` that takes two arguments, an association list, `alist`, and an associated datum `val`, and returns

- an element from `alist` that has `val` as its second component, if such an element exists
- `#f` if there is no such element.

In solving this problem, you may not use the generic `lookup` procedure from the reading.

c. Define and test a procedure that takes two parameters, an association list, `alist`, and an associated datum, `val`, and returns a list of all elements that have `val` as the second component.

## Exercise 8: Using a Specific Database

For some problems, it seems natural to always use a specific database, rather than to pass the database as a parameter. For example, suppose we'd set up a table of science department chairs (which may sound familiar from the reading, although we've expressed it differently here).

```
;;; Value:
;;; science-department-chairs
;;; Type:
;;; List of lists.
;;; Each sublist is of length two and contains a department (or "science")
;;; and a name.
;;; Both of those values are strings.
;;; Contents:
;;; A list of the department and division chairs in the Science division
;;; in academic year 2006-2007.
(define science-department-chairs
  (list (list "Science" "Samuel A. Rebelsky" "4410")
        (list "Biology" "Vince Eckhart" "4354")
        (list "Chemistry" "Lee Sharpe" "3008")
        (list "Computer Science" "Henry Walker" "4208")
        (list "CS" "Henry Walker" "4208")
        (list "Mathematics and Statistics" "Marc Chamberland" "4207")
        (list "Mathematics" "Marc Chamberland" "4207")
        (list "Math/Stats" "Marc Chamberland" "4207")
        (list "Math" "Marc Chamberland" "4207")
        (list "Physics" "Bob Cadmus" "3016")
        (list "Psychology" "Janet Gibson" "3168")
        (list "Psych" "Janet Gibson" "3168")
        (list "Statistics" "Marc Chamberland" "4207")))
```

We can write a procedure to look up a department chair as follows:

```
;;; Procedure:
;;; lookup-science-chair
;;; Parameters:
;;; dept, the name of a science department (or simply "Science")
;;; Purpose:
;;; Look up the chair of a science department.
;;; Produces:
;;; chair, a string (or #f)
;;; Preconditions:
;;; science-department-chairs must be defined appropriately
;;; dept must be a string
;;; Postconditions:
;;; If science-department-chairs specifies a chair for dept,
;;; chair is that chair.
;;; Otherwise, chair is false (#f)
(define lookup-science-chair
  (lambda (dept)
    (assoc dept science-department-chairs)))
```

The strategy of using a specific database in a procedure is often called *hard-coding* the database.

- a. Using `lookup-science-chair`, look up the chair of this department.
- b. Using `lookup-science-chair`, look up the chair of Geology.
- c. Suppose we wanted to write the converse procedure (one that given a name, tells which department he or she chairs). Can we still hard-code the database? If so, show how. If not, explain why not.

## For Those with Extra Time

### Extra 1: Lists as Keys

a. Define and test a procedure, (`weird-lookup sym weirdlists`), that takes two arguments, the first an atom and the second an association list whose keys are all lists of atoms. The procedure should return a list of all the values whose keys contain the atom. For example,

```
> (define whatever
  (list
    (list (list 'a 'b 'c) 'ABCs)
    (list (list 'a 'e 'i 'o 'u) 'vowels)
    (list (list 'e 't 'a 'i 'o 'n) 'shrdlu)))
> (weird-lookup 'a whatever)
(abcs vowels shrdlu)
> (weird-lookup 'b whatever)
(abcs)
> (weird-lookup 'o whatever)
(vowels shrdlu)
> (weird-lookup 'q whatever)
()
```

Note that you might find the `member` procedure helpful.

b. Why might you want to use this procedure?

### Extra 2: Compound Keys

Consider the following list of faculty in Mathematics, Computer Science, and Statistics.

```
(define csms
  (list
    (list "Sam" "Rebelsky" "Associate" "Computer Science")
    (list "Henry" "Walker" "Chaired" "Computer Science")
    (list "John" "Stone" "Instructor" "Computer Science")
    (list "Janet" "Davis" "Assistant" "Computer Science")
    (list "Marge" "Coahran" "Visitor" "Computer Science")
    (list "Emily" "Moore" "Full" "Mathematics")
    (list "Karen" "Shuman" "Assistant" "Mathematics")
    (list "Keri" "Kornelson" "Assistant" "Mathematics")
    (list "Holly" "Mosley" "Visitor" "Mathematics")
    (list "David" "Romano" "Assistant" "Mathematics")
    (list "Marc" "Chamberland" "Chair" "Mathematics"))
```

```
(list "Royce" "Wolf" "Associate" "Mathematics")
(list "Gene" "Herman" "SFS" "Mathematics")
(list "Chuck" "Jepsen" "SFS" "Mathematics")
(list "Arnie" "Adelberg" "SFS" "Mathematics")
(list "Tom" "Moore" "Full" "Statistics")
(list "Shonda" "Kuiper" "Assistant" "Statistics")
(list "Leah" "Jager" "Visitor" "Statistics"))
```

When looking up people, we might want to make sure that we match both first and last names. Write a procedure, (`lookup first last people`) that extracts the entry in which both *first* and *last* match.

## Notes

Here you will find notes on selected problems.

### Notes on Exercise 3

Note: The value of `sidekicks` is not a procedure, so it is not necessary to use a lambda-expression in this exercise. Look at the definition of `science-chairs-directory` for an example of the form that your definition of `sidekicks` should take.

---

Copyright © 2006 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.