Lab: Association Lists

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

- **Exercises**
  - Exercise 0: Preparation
  - Exercise 1: Birthdays
  - Exercise 2: Finding Birthdays
  - Exercise 3: Duplicate Keys
  - Exercise 4: Preconditions
  - Exercise 5: Reverse Associations
  - Exercise 6: Using a Specific Database
  - Exercise 7: Compound Keys
- **Notes**
  - Note on exercise 1

**Exercises**

**Exercise 0: Preparation**

a. Make sure you know that the \texttt{assoc} procedure does.

b. Start DrScheme

**Exercise 1: Birthdays**

Define an association list \texttt{birth-dates} that associates the surnames of recent presidents of the United States (as strings) with their birth-dates (again, as strings).

You may also want to look at the \texttt{note on this exercise}

Here’s a table containing information for your association list:
### Exercise 2: Finding Birthdays

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

### Exercise 3: Duplicate Keys

a. Redefine `birth-dates` so that it includes two entries with the same key, for two people who have the same surname -- say, John Adams (born October 30, 1735) and John Quincy Adams (born July 11, 1767). What happens if you try to apply `assoc` to retrieve these entries, using the common key "Adams"?

b. Many people find these results disappointing. To help alleviate this disappointment, define and test a procedure similar to `assoc`, except that it returns a list of all the pairs with the given key.

### Exercise 4: 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 last name of a faculty member?

<table>
<thead>
<tr>
<th>President</th>
<th>Date of birth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clinton</td>
<td>August 19, 1946</td>
</tr>
<tr>
<td>Bush</td>
<td>June 12, 1924</td>
</tr>
<tr>
<td>Reagan</td>
<td>February 6, 1911</td>
</tr>
<tr>
<td>Carter</td>
<td>October 1, 1924</td>
</tr>
<tr>
<td>Ford</td>
<td>July 14, 1913</td>
</tr>
<tr>
<td>Nixon</td>
<td>January 9, 1913</td>
</tr>
<tr>
<td>Johnson</td>
<td>August 27, 1908</td>
</tr>
<tr>
<td>Kennedy</td>
<td>May 29, 1917</td>
</tr>
<tr>
<td>Eisenhower</td>
<td>October 14, 1890</td>
</tr>
</tbody>
</table>
b. Confirm or refute your answers by experimentation.

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

**Exercise 5: Reverse Associations**

a. What happens if you search by date instead of by person? For example, you might try
(assoc "October 1, 1924" birth-dates).

b. Define and test a procedure **reverse-lookup** 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.

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 6: 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).

```scheme
;;; 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
(define science-department-chairs
  (list (list "Science" "Bruce Voyles")
    (list "Biochemistry" "Bruce Voyles") ; Well ...
    (list "Biology" "Diane Robertson")
    (list "Chemistry" "Lee Sharpe")
    (list "Math/CS" "Emily Moore")

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

```scheme
;;; Procedure:
;;;   look-up-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, if the department has a chair
;;;   #f, otherwise
;;; Preconditions:
;;;   science-department-chairs must be defined appropriately
;;;   dept must be a string
;;; Postconditions:
;;;   If the procedure returns a string, chair is the chair of dept.
;;;   If the procedure fails to return a string, that department has
;;;     no chair (or isn’t even a department).
(define look-up-science-chair
  (lambda (dept)
    (if (assoc dept science-department-chairs)
      (cadr (assoc dept science-department-chairs))
      #f)))
```

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

a. Using `look-up-science-chair`, look up the chair of this department.

b. Using `look-up-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.

**Exercise 7: Compound Keys**

a. Define and test a procedure 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.

b. Why might you want to use this procedure?

**Notes**
Note on exercise 1

Note: The value of birth-dates 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 birth-dates should take.