

## LispCourse #26: Reprise; Continuation of Record Package from #25

### Reprise: Sample Programming Exercise

#### The Problem

Consider an existing database containing the membership roster for the ACM.

The database is a list of entries.

Each entry corresponds to a person and contains the person's name, a list of dates relevant to that person's membership, a list of special interest groups(SIGs) that person belongs to, and an atom indicating whether the person's membership status.

The person's name is a list of three atoms corresponding to the persons first, middle, and last names in that order.

The date list contains two dates, the date of the person joined ACM and the date of the last renewal.

Each date is a list of three numbers: day, month, and year.

The list of SIGs consists of any number (including 0) of atoms like SIGOA and SIGART.

The membership status is one of the following atoms: Member, Associate, or Student.

Example Database:

```
((JoAnn A Jones)((23 10 75)(01 10 84))(SIGCHI SIGART SIGED)
Member)
((Bill B Smith)((12 03 82)(01 04 85))(SIGOA) Associate)
((Jill Jane Jerzy)((15 09 80)(14 09 84)) NIL Student))
```

Write functions to:

1. Add a new person entry to this database.
2. Return a list of all SIGCHI members in the ACM.
3. Given a last name, update that person's membership status to Member.
4. Given a last name and a date, update that person's last renewal date.

You can assume that the database is the value of the atom ACM.Members.

### **Step 1: Describe the database**

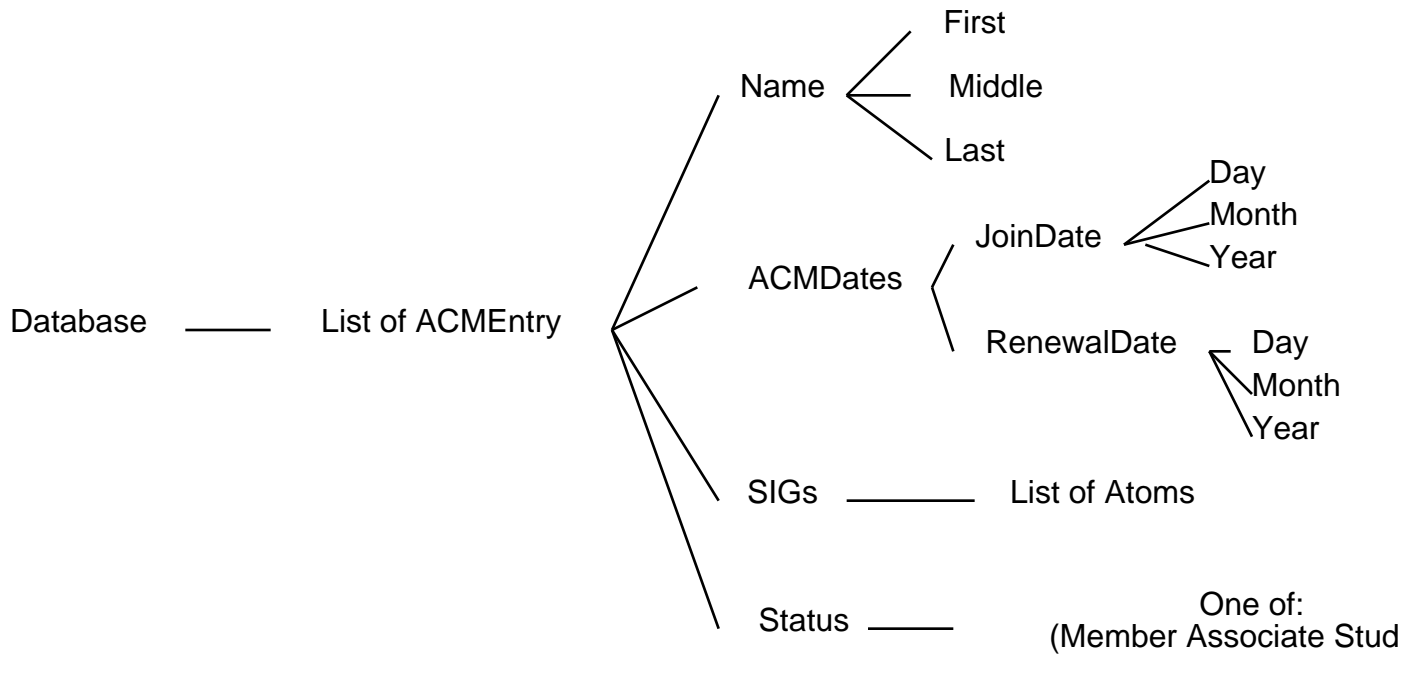
(RECORD ACMEntry (Name Dates SIGs Status))

(RECORD Name (First Middle Last))

(RECORD ACMDates (JoinDate RenewalDate))

(RECORD Date (Day Month Year))

Note: The whole database and the SIGs list are variable length and therefore cannot be described as RECORDs. In fact, they are likely to be the kind of data structures that you search looking for something rather than the kind of data structures you want to get a particular part (i.e., field) of.



ent)

## Step 2: Write the constructors, selectors, and mutators for the data structures

A. Start with the lowest level structures (i.e., the one that consist of atoms)

i. Names: Deal with the Name RECORD

```

(DEFINEQ
  (ACM.CreateName
    (LAMBDA (First Middle Last)
      (CREATE Name First _ First Middle _ Middle last _
        Last)))
  (ACM.FirstName
    (LAMBDA (OldName NewFirst)
      (COND
        (NewFirst (replace (Name First) of OldName with
          NewFirst))
        (T (fetch (Name First) of OldName)))))
  (ACM.MiddleName
    (LAMBDA (OldName NewMiddle)
      (COND

```

```

                                (NewMiddle (replace (Name Middle) of OldName
                                with NewMiddle))
                                (T (fetch (Name Middle) of OldName))))))
(ACM.LastName
  (LAMBDA (OldName NewLast)
    (COND
      (NewLast (replace (Name Last) of OldName with
                        NewLast))
      (T (fetch (Name Last) of OldName))))))

```

ii. Dates: Deal with the Date RECORD

```

(DEFINEQ
  (ACM.CreateDate
    (LAMBDA (Day Month Year)
      (CREATE Date Day _ Day Month _ Month Year _ Year)))
  (ACM.Day
    (LAMBDA (OldDate NewDay)
      (COND
        (NewDay (replace (Date Day) of OldDate with
                          NewDay))
        (T (fetch (Date Day) of OldDate))))))
  (ACM.Month
    (LAMBDA (OldDate NewMonth)
      (COND
        (NewMonth (replace (Date Month) of OldDate with
                           NewMonth))
        (T (fetch (Date Month) of OldDate))))))
  (ACM.Year
    (LAMBDA (OldDate NewYear)
      (COND
        (NewYear (replace (Date Year) of OldDate with
                           NewYear))
        (T (fetch (Date Year) of OldDate))))))

```

B. Continue with next level structures: the ones that use Names and Dates

## i. ACMDates

```

(DEFINEQ
  (ACM.CreateACMDateList
    (LAMBDA (JoinDate RenewalDate)
      (CREATE ACMDates JoinDate _ JoinDate RenewalDate _
        RenewalDate)))
  (ACM.JoinDate
    (LAMBDA (DatesList NewJoinDate)
      (COND
        (NewJoinDate (replace (ACMDates JoinDate) of
          DatesList with NewJoinDate))
        (T (fetch (ACMDates JoinDate) of DatesList)))))
  (ACM.RenewalDate
    (LAMBDA (DatesList NewRenewalDate)
      (COND
        (NewRenewalDate (replace (ACMDates
          RenewalDate) of DatesList with
          NewRenewalDate))
        (T (fetch (ACMDates RenewalDate) of
          DatesList)))))

```

## C. Highest level record structure: the ACMEntry

```

(DEFINEQ
  (ACM.CreateACMEntry
    (LAMBDA (Name Dates SIGs Status)
      (CREATE ACMEntry Name _ Name Dates _ Dates SIGs _ SIGs
        Status _ Status)))
  (ACM.Name
    (LAMBDA (Entry NewName)
      (COND
        (NewName (replace (ACMEntry Name) of Entry
          with NewName))
        (T (fetch (ACMEntry Name) of Entry)))))
  (ACM.DatesList

```

```

(LAMBDA (Entry NewDatesList)
  (COND
    (NewDatesList (replace (ACMEntry Dates) of
      Entry with NewDatesList))
    (T (fetch (ACMEntry Dates) of Entry))))))
(ACM.SIGs
  (LAMBDA (Entry NewSIGs)
    (COND
      (NewSIGs (replace (ACMEntry SIGs) of Entry with
        NewSIGs))
      (T (fetch (ACMEntry SIGs) of Entry))))))
(ACM.Status
  (LAMBDA (Entry NewStatus)
    (COND
      (NewStatus (replace (ACMEntry Status) of Entry
        with NewStatus))
      (T (fetch (ACMEntry Status) of Entry))))))

```

**Step 3: Write the functions that do the work as required.**

1. Add a new person entry to this database.

*Plan: Create an entry record for the new person and CONS it onto the existing database list.*

```

(DEFINEQ
  (ACM.AddPerson
    (LAMBDA (First Middle Last JoinDate RenewalDate SIGs Status)
      (SETQ ACM.Members
        (CONS
          (ACM.CreateACMEntry
            (ACM.CreateName First Middle
              Last)
            (ACM.CreateACMDatesList
              JoinDate
                RenewalDate)

```

```

      SIGs Status)
    ACM.Members))))))

```

2. Return a list of all SIGCHI members in the ACM.

*Plan: Iterate through list of entries. For each entry that is a member of SIGCHI, add it to a list of the SIGCHI members (which is the value of the atom SIGCHI.Members).*

```

(DEFINEQ
  (ACM.SIGCHIMembers
    (LAMBDA NIL
      (SETQ SIGCHI.Members NIL)
      (FOR Entry in ACM.Members DO
        (COND
          ( (MEMBER 'SIGCHI (ACM.SIGS Entry))
            (SETQ SIGCHI.Members
              (CONS Entry
                SIGCHI.Members))))
          (T
            (SETQ SIGCHI.Members))))
    SIGCHI.Members)))

```

3. Given a last name, update that person's membership status to Member.

*Plan: Iterate through list of entries. For the entry that matches on last name, change the Status field.*

```
(DEFINEQ
  (ACM.ChangeStatusToMember
    (LAMBDA (LastName)
      (FOR Entry in ACM.Members DO
        (COND
          ( (EQUAL LastName
                    (ACM.LatName (ACM.Name
                                   Entry)))
            (ACM.Status Entry 'Member)))))))
```

4. Given a last name and a date, update that person's last renewal date.

*Plan: Iterate through list of entries. For the entry that matches on last name, change the renewal date.*

```
(DEFINEQ
  (ACM.ChangeRenewalDate
    (LAMBDA (LastName NewRenewalDate)
      (FOR Entry in ACM.Members DO
        (COND
          ( (EQUAL LastName
                    (ACM.LatName (ACM.Name
                                   Entry)))
            (ACM.RenewalDate
              (ACM.DatesList Entry)
              NewRenewalDate)))))))
```



**Step 4: Save your work on a file and list it.**

Decide to call the file ACMLIST.

Set up the COMS list for MAKEFILE

```
(SETQ ACMLISTCOMS (QUOTE (
  (RECORDS ACMEntry Name ACMDates Date)
  (VARS ACM.Members)
  (* * Constructors, Selectors, and Mutators)
  (FNS ACM.CreateName ACM.FirstName ACM.MiddleName
    ACM.LastName)
  (FNS ACM.CreateDate ACM.Day ACM.Month ACM.Year)
  (FNS ACM.CreateACMDateList ACM.JoinDate
    ACM.RenewalDate)
  (FNS ACM.CreateACMEntry ACM.Name ACM.DatesList
    ACM.SIGs ACM.Status)
  (* * Work-doing functions)
  (FNS ACM.AddPerson ACM.SIGCHIMembers
    ACM.ChangeStatusToMember ACM.ChangeRenewalDate)
)))
```

Do the MAKEFILE

```
(MAKEFILE '{PHYLUM}<HALASZ>LISPCOURSE>ACMLIST)
```

Do the LISTFILES

```
(LISTFILES {PHYLUM}<HALASZ>LISPCOURSE>ACMLIST)
```

**Step 5: Test, Debug, and redo the MAKEFILE.**

EFS.

## Continuation from #25: Editing Record/Datatype Declarations and Instances

### Examining and Editing Record/Datatype Declarations: RECLOOK and EDITREC

To see the declaration for record or datatype named *RecordName*, use **(RECLOOK *RecordName*)**.

```
61_ (RECORD ACMDDate (Month Day Year) Month _ 01 Day _ 01 Year
    _ 00)
```

*ACMDDate*

```
62_ (RECLOOK 'ACMDDate)
```

*(RECORD ACMDDate (Month Day Year) Month \_ 01 Day \_ 01 Year \_ 00)*

```
63_ (DATATYPE ACMDDate (Month Day Year))
```

*(record ACMDDate redeclared)*

*ACMDDate*

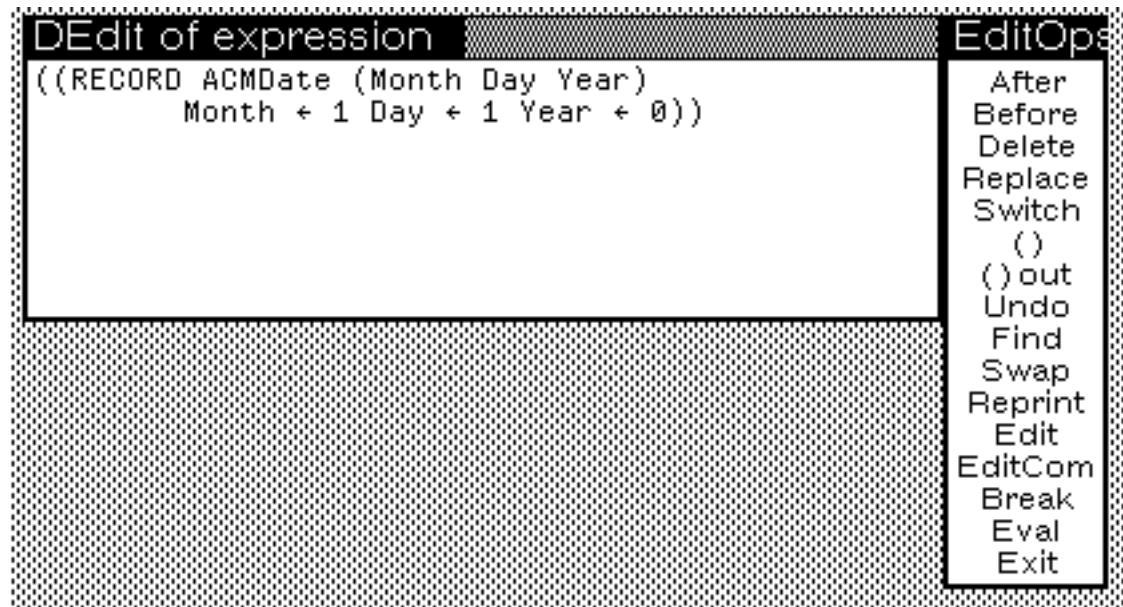
```
64_ (RECLOOK 'ACMDDate)
```

*(DATATYPE ACMDDate (Month Day Year))*

To edit (i.e., change) the declaration for record or datatype named *RecordName*, use **(EDITREC *RecordName*)**. [EDITREC is an NLAMBDA function]

This will pop you into a DEdit on the record/datatype declaration. If you exit DEdit with **OK**, the record/datatype will be redeclared. If you exit with **Stop**, no change in the declaration will happen.

**(EDITREC ACMDDate)**



### Examining and Editing Record/Datatype Instances: The Inspector

To examine or change the value of a field in a record/datatype instance, use the Inspector. The Inspector is called as follows:

**(INSPECT *Instance*)**

If *Instance* is a *record*, then a menu will be displayed with the following choices: *DisplayEdit*, *TtyEdit*, *Inspect*, *AsRecord*.

Choose: *AsRecord*

This will bring up a menu of all the RECORD types in the system. Choose the appropriate record type from this menu (e.g., choose ACMDate if the record is an ASCMDate).

Finally, this brings up an Inspect window (after prompting for a location) displaying the fields and values of the fields for the given *Instance* of the chosen record type.

If *Instance* is a *datatype*, then an Inspect window will be displayed immediately (after prompting for a location) showing the fields and values of the fields for the given *Instance*.

An Inspect window looks as follows:

```
75_ (RECORD ACMDate (Month Day Year) Month _ 01 Day _ 01 Year _ 00)
```

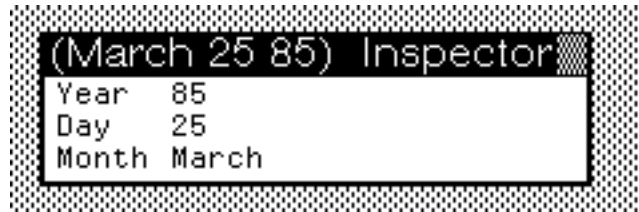
```
ACMDate
```

```
76_ (SETQ Test (CREATE ACMDate Month _ March Year _ 85 Day _ 25))
```

```
(March 25 1985)
```

```
77_ (INSPECT Test)
```

```
{WINDOW}#33,123456
```

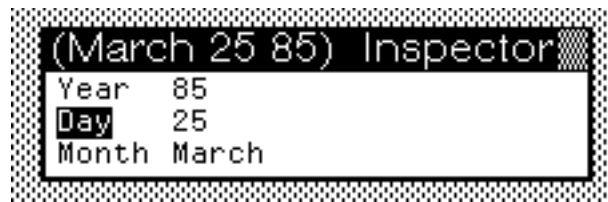


The Inspect window has two columns:

The Left-hand column lists the *field names* of the ACMDate record type.

The Right-hand column lists the corresponding *values* of these fields in the instance being inspected.

You can select any item in the Inspect window by pointing to it and clicking the **LEFT** mouse button. The selected item will be inverted.



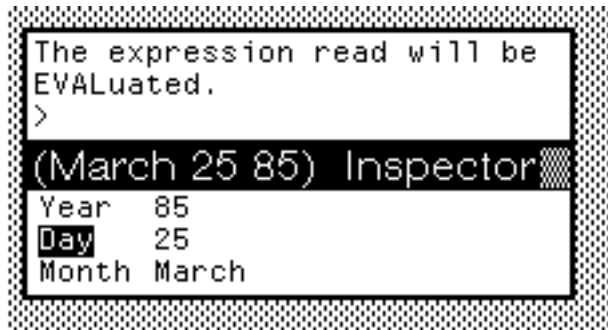
You can *act* on the selected item by clicking the **MIDDLE** mouse button anywhere inside the Inspect window. The action taken depends on whether the selected item is a field (left column) or a value (right column).

If the selected item is a *field*: You can change the *value* for that field as follows:

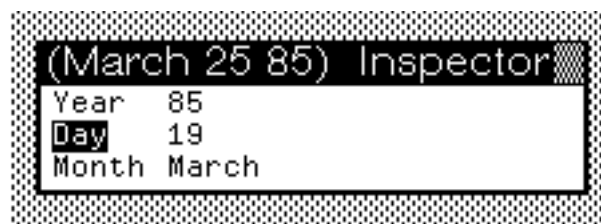
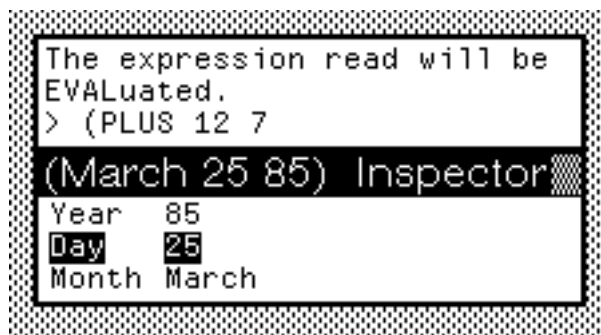
Clicking the **MIDDLE** button will bring up a menu of one item:  
*Set*

Choose this item. (selecting off the menu aborts the interchange)

This will bring up a small prompt box above the Inspect window.

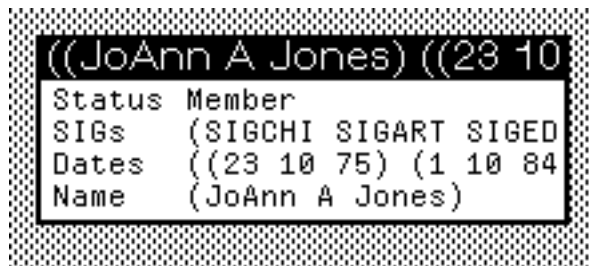


Type an S-expression that will evaluate to the new value you wish the field to have. This expression will be evaluated and the value of the selected field will be set to the result.

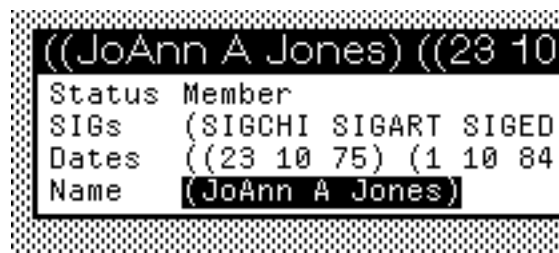


If the selected item is a *value*: (INSPECT value) will be evaluated. This is useful for Inspecting embedded record structures.

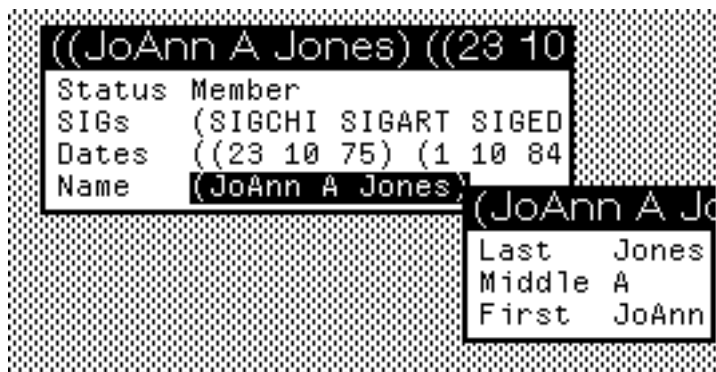
Example: Inspecting an ACMEEntry instance



Selecting the **value** of the Name field using the LEFT button



Clicking the MIDDLE button to INSPECT the name value.



Note: the interaction sequence to bring up the second inspector is the same as for the first inspector, i.e., I had to specify *AsRecord* and choose the *Name* record type from the menu of all record types.

To get rid of an Inspect window ž just close it using the RIGHT button menu.

### Using the Inspector

The inspector is very useful to examine your data structures in an interpretable way while you are debugging.

For example:

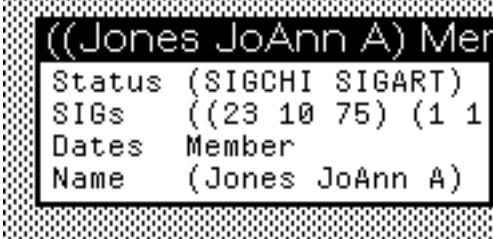
Consider the RECORD definitions given in the ACM problem above.

Consider the sample database:

```
(SETQ ACM.Members
  '(((Jones JoAnn A ) Member ((23 10 75)(01 10 84))(SIGCHI
    SIGART))
    ((Smith Bill B ) Associate ((12 03 82)(01 04 85))(SIGOA))
    ((Jerzy Jill Jane ) Student ((15 09 80)(14 09 84)) NIL)))
```

None of our functions we defined above would work on this sample database. Why? A quick inspect shows why.

1. Inspect the first entry: (INSPECT (CAR ACM.Members))



```
((Jones JoAnn A) Mer
Status (SIGCHI SIGART)
SIGs ((23 10 75) (1 1)
Dates Member
Name (Jones JoAnn A)
```

Notice that the fields and values don't quite match up; e.g., Status field is a list.

2. Notice that Name doesn't look quite right. So Inspect it.



```
((Jones JoAnn A) Mer
Status (SIGCHI SIGART)
SIGs ((23 10 75) (1 1)
Dates Member
Name (Jones JoAnn A)
```

((Jones JoAnn A) Men	
Status	(SIGCHI SIGART)
SIGs	((23 10 75) (1 1
Dates	Member
Name	(Jones JoAnn A)

(Jones JoAn	
Last	A
Middle	JoAnn
First	Jones

Notice that the fields and values don't quite match up in the name either; e.g., Last name field is a middle initial.

Bottom line: Either the sample database is in error or our RECORD declarations are in error. In this case, its the former.

## References

The Inspector is described in Section 20.4 of the IRM.

EDITREC and RECLOOK are in Section 3.7 of the IRM.

## Exercise

For the following exercise, it might be handy to know about the following embellishment of the FOR loop:

**(FOR *Variable* IN *List* WHEN *Predicate* DO *S-Exprs*)**

Instead of executing the *S-Exprs* following the DO on every iteration, this FOR loop evaluates the *Predicate* following the WHEN on every iteration. If *Predicate* evaluates to a non-NIL value, then the *S-exprs* following DO are evaluated. If *Predicate* evaluates to NIL, then the FOR loop skips the *S-exprs* after the DO on this iteration.

Note that the WHEN clause works with COLLECT as well as DO.

Examples:

```
76_(FOR A IN '((T 1)(NIL 2)(T 3)(NIL 4))
      WHEN (CAR A) COLLECT (CADR A))
(1 3)
```



```
77_(FOR A IN '((T 1)(NIL 2)(T 3)(NIL 4))  
      WHEN (NULL (CAR A)) COLLECT (CDR A))  
((2) (4))
```

**Exercise from Touretzky, p. 179. (See following few pages)**