Week 02 - Lecture 2 Slides

Dr. Yeganeh Bahoo

Created: 2023-09-24 Sun 21:09

Table of Contents

Lecture 2: Lisp crash course (cont.)

Learning objectives

By the end of this lecture you should be able to:

- Write lisp programs using DO and DO* loops.

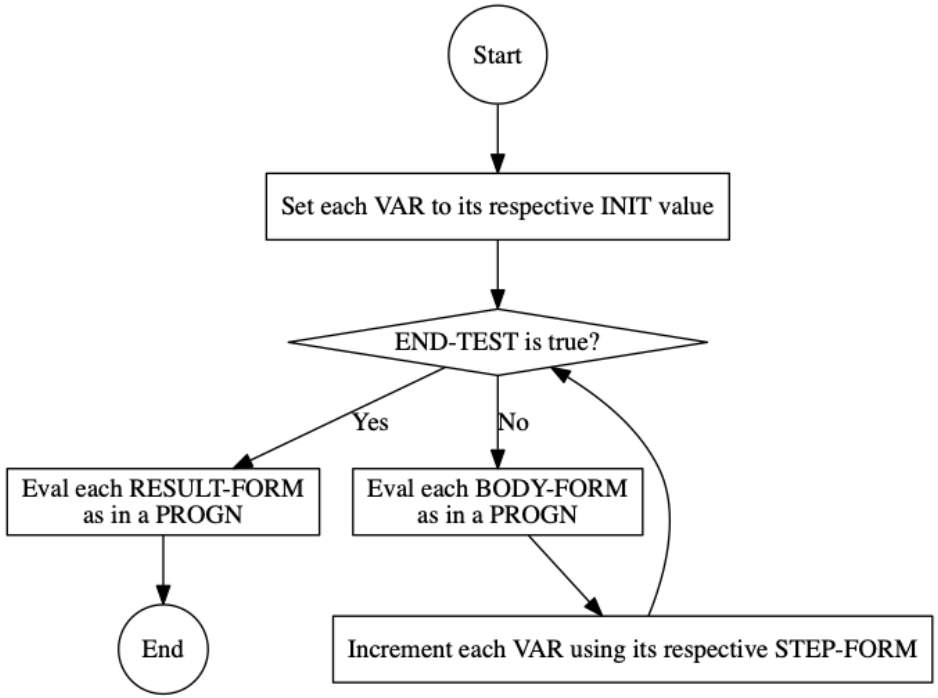
looping: DO and DO*

Syntax & Semantics

```
(DO (var-definition*)
    (end-test result-form*)
```

```
body-form*)
```

- each *var-definition* is a list $(var\ init\ [step-form])$
- remember the notation we have been using to represent lisp syntax:
  - $form^*$ means "zero or more *form*"
  - $[form]$ means "*form* is optional".



Example:

```
RTL-USER> (do ((temp1 1 (1+ temp1))  ; var, init, and step
               (temp2 0 (1- temp2))) ; var, init, and step
              ((> (- temp1 temp2) 5) temp1) ; end-test and return
            (format t "temp1: ~a temp2: ~a~%" temp1 temp2)) ; body
temp1: 1 temp2: 0
temp1: 2 temp2: -1
temp1: 3 temp2: -2
4
```

DO and DO* (cont.)

- As in LET vs LET*, in a DO all initialization forms are evaluated before their values are bound to the variables
- As such, the form `(length s)` in line 5 would cause an error in a DO because there is no value bound to *s* when that form is evaluated.
- That's why we need to use a DO* instead

```
(defun do*-example ()                       ; 1
  (format t "Type 'exit' to stop.~%")       ; 2
  (do* ((i 0 (1+ i))                        ; 3
        (s (read-line) (read-line))         ; 4
```

```
        (acc (length s) (+ acc (length s)))) ; 5
       ((equal s "exit") t)
    (format t "~a: ~a ~a~%" i s acc)))
```

```
CL-USER> (do*-example)
 Type 'exit' to stop.
 test
 0: test 4
 abc
 1: abc 7
 def
 2: def 10
 exit
 T
```

Exercise

Complete the blanks.

```
RTL-USER> (count-char #\a "abba")
2
```

```
(defun count-chars (c s)
  "Returns the count of characters that are equal to c in string s"
  (let ((acc 0))
    (dotimes (i ... ...)
      (when (char= ... c)
        (incf acc)))))  ; equivalent to (setf acc (+ 1 acc))
```

For further study: read the documentation on INCF

Solution

```
(defun count-chars (c s)
  (let ((acc 0))
    (dotimes (i (length s) acc)
      (when (char= (aref s i) c)
        (incf acc)))))  ; equivalent to (setf acc (+ 1 acc))
```

Exercise

Rewrite function COUNT-CHAR using DO instead of DOTIMES.

```
(defun count-char (c s)
  (let ((acc 0))
    (dotimes (i (length s) acc)
      (when (char= (aref s i) c)
        (incf acc)))))  ; equivalent to (setf acc (+ 1 acc))
```

```
(defun count-char (c s)
  (do ((i 0 ...)
       (acc 0))
      ((equal i ...) ...)
      (when (char= (aref s i) c)
        (incf acc)))))  ; equivalent to (setf acc (+ 1 acc))
```

Solution

```
(defun count-char (c s)
  (do ((i 0 (1+ i))
       (acc 0))
      ((= i (length s)) acc)
      (when (char= (aref s i) c)
        (incf acc)))))  ; equivalent to (setf acc (+ 1 acc))
```

Exercise

Given the ASCII table and the definition of VOWELP below, write a definition for CONSONANTP

```
(defun vowelp (c)
  "Returns true if c is a character representing a vowel"
   (or (char= c #\a) (char= c #\A)
       (char= c #\e) (char= c #\E)
       (char= c #\i) (char= c #\I)
       (char= c #\o) (char= c #\O)
       (char= c #\u) (char= c #\U)))
```

```
(consonantp #\a)  =>  NIL

(consonantp #\@)  => NIL

(consonantp #\c)  =>  T
```

```
(defun consonantp (c)
  (and (not (vowelp c))
       ...
```

Solution

```
(defun consonantp (c)
  (and (not (vowelp c))
       (or (and (char>= c #\a) (char<= c #\z))
           (and (char>= c #\A) (char<= c #\Z)))))
```

Exercise

Rewrite function COUNT-VC below using a DO loop. This function returns the number of vowels and consonants of a string, in that order

```
CL-USER> (count-vc "a casa")
3
2
```

```
(defun count-vc (s)
  (let ((accv 0)
        (accc 0))
    (dotimes (i (length s) (values accv accc))
      (if (vowelp (aref s i)) (incf accv)
          (if (consonantp (aref s i)) (incf accc))))))
```

Solution

```
(defun count-vc-do (s)
  (do ((i 0 (1+ i))
       (accv 0)
       (accc 0))
      ((equal i (length s)) (values accv accc))
      (if (vowelp (aref s i)) (incf accv)
          (if (consonantp (aref s i)) (incf accc)))))
```