CS151.01 2009F *Functional Problem Solving*

Class 20: Naming Local Values

Held: Wednesday, 30 September 2009

Summary: We consider why and how you name values within a procedure, and limit access to those names to within the procedure.

Related Pages:

- EBoard.
- Lab: Local Bindings.
- Reading: Local Bindings.
- Due: Assignment 4: Exploring Colors.

Notes:

- I will not have office hours today. Sorry.
- Please keep your partners from Monday.
- Assignment 5 is now ready.
- EC for attending some part of the Rosenfield anniversary symposium on Thursday.
- Don't forget about the history of programming languages talk at noon on Friday in 3821.
- Reading for Friday: Boolean Values and Predicate Procedures.

Overview:

- Why name things.
- Naming things with `let`.
- Naming things with `let*`.
- Naming procedures.
- Lab.

The Problem: Naming Values

- As we've seen in many problems, it helps to name the values that we use within our procedure. Why?
  - It can make the code more readable because the name tells us something about the role the value plays.
  - It can make the code more efficient, because it allows us to avoid recomputing a value.
- Another reasons to name things is that we might want to create helper procedures and only make them available to the current procedure.

Naming Things with `let`

- You name things with `let`.
- `let` has the form

```
(let ((name_1  exp_1 )
      (name_2  exp_2 )
      ...
      (name_n  exp_n ))
  body)
```

- `let` has the meaning:
  - Evaluate all the expressions.
  - Update the binding table to associate each name with the corresponding value.
  - Evaluate *body* using the updated binding table.
  - Eliminate all the bindings just created.
- You can use `let` in a simple expression:

```
(define values (list 1 4 2 4 1 5 9))
(let ((largest (max values))
      (smallest (min values)))
  (/ (+ largest smallest) 2))
```

- More frequently, we use `let` within a procedure.

Sequencing Bindings with `let*`

- If we want to bind some things in sequence, we need to use `let*` rather than `let`.
- `let*` has the form

```
(let* ((name_1  exp_1 )
       (name_2  exp_2 )
       ...
       (name_n  exp_n ))
  body)
```

- `let*` has the meaning:
  - Evaluate exp_1.
  - Update the binding table to associate name_1 with that value.
  - Evaluate exp_2.
  - Update the binding table to associate name_2 with that value.
  - ...
  - Evaluate exp_n.
  - Update the binding table to associate name_n with that value.
  - Evaluate *body* using the updated binding table.
  - Eliminate all the bindings just created.