

Class 12: Recursion, Revisited

Held: Friday, February 9, 2007

Summary: Today we continue our consideration of the process of recursion.

Related Pages:

- EBoard.
- Lab: Recursion.
- Reading: Recursion.

Due

- HW 6

Notes:

- Amazingly enough, the reading for Monday is the same as the reading for today. (The goal is that you move from reading procedures to writing them.)
- Yes, I do intend to get grading done this weekend. I apologize for the delay.

Overview:

- Q&A.
- Lab.
- Optional topics: Scheme's Evaluation Strategy; Another Example.

Questions and Answers

- I will reserve time for questions and answers.

Lab

- Continue to work on the lab.

Some Extra Stuff

I included the following notes in previous offerings of this course. I will cover them if (or when) questions suggest that such coverage is necessary.

Detour: Scheme's Evaluation Strategy

- Since it came up yesterday (and has come up indirectly in the past), let's revisit (or perhaps visit) the strategy Scheme uses for evaluating expressions.
- If the expression to be evaluated is a name, look up the value associated with the name and then evaluate the value.
- If the expression to be evaluated is a primitive value (a number, string, character, procedure, boolean, symbol, or some other type that we haven't yet covered), just use that value.
- If the expression to be evaluated is a procedure application (typically indicated by an open paren), evaluate all of the parameters to the procedure and then apply the procedure.
 - We'll come back to how you apply procedures.
- If the expression to be evaluated is a special operation (also called a *macro*), such as `and`, `or`, `if`, and `cond`, follow the evaluation rules for that operation.
- If the procedure to be applied is a user-defined procedure, substitute the actual parameters (those that appear in the procedure application) for the formal parameters (those that appear in the procedure definition) in the body, and then evaluate the body.
- If the procedure to be applied is a built-in procedure, follow the rules for the built-in procedure.
- For example, given, consider the following definitions:

```
(define double
  (lambda (x)
    (+ x x)))
(define a 5)
```

- Consider the following expression
`(double (double (* 7 (+ 1 a))))`
- The top-level expression is a procedure application, so we need to evaluate the parameters.
- The only parameter is `(double (* 7 (+ 1 a)))`
- Again, this is a procedure application, so we evaluate its parameters.
- The only parameter is `(* 7 (+ 1 a))`.
- Again, this is a procedure application, so we evaluate its parameters.
- There are two parameters, 7 and `(+ 1 a)`.
- 7 is a primitive value, so we're done with it.
- `(+ 1 a)` is a procedure application, so we evaluate its parameters.
- There are two parameters, 1 and `a`.
- 1 is a primitive value, so we're done with it.
- `a` is the name for 5, so we use that.
- We're now ready to compute `(+ 1 5)`. Since `+` is a built-in operation, we do what it's supposed to do, and the value of this sub-expression is 6.
- We're now ready to compute `(* 7 6)`. Again, we rely on the built-in operation, and get 42 for this subexpression.
- We're now ready to compute `(double 42)`. This time, we have a user-defined operation, so we plug in 42 for `x` in the body, and get `(+ 42 42)`.
- The value of that sum is 84.

- We're now ready to compute `(double 84)`. This is left as an exercise for the reader.

Another Example: Difference

- To help understand the difference between the two versions of `SUM`, let us consider a variant in which we compute a similar value, but using subtraction rather than addition.

```
(define difference
  (lambda (numbers)
    (if (null? numbers)
        0
        (- (car numbers) (difference (cdr numbers))))))

(define new-difference
  (lambda (numbers)
    (new-difference-helper (car numbers) (cdr numbers))))

(define new-difference-helper
  (lambda (difference-so-far remaining)
    (if (null? remaining)
        difference-so-far
        (new-difference-helper (- difference-so-far (car remaining))
                                (cdr remaining)))))
```

- Do we get the same values from these two procedures? Why or why not?

Copyright © 2007 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.