

Class 11: Recursion

Held: Wednesday, February 7, 2007

Summary: Today we begin to consider one of the most powerful tools in Scheme, *recursion*. Recursion allows you to repeat operations.

Related Pages:

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

Notes:

- Are there questions on HW6?
- I got the topic of tomorrow's Thursday Extra wrong - It's on the automation of the Athletic Recruiting Practice.
- For Friday, please reread the reading on recursion.

Overview:

- Repetition.
- Recursion.
- Recursion in Scheme.
- Lab.

Repetition

- You may recall that when we first considered algorithms we identified a number of key aspects of algorithms:
 - *variables*: the ability to name things;
 - *conditions*: the ability to choose between things;
 - *procedures*: the ability to name (and parameterize) collections of steps;
 - *repetition*: the ability to do something multiple times;
 - *input* and *output*: the ability to get and report values.
- We've already seen how to do almost all of these things, except for repetition.
- Examples of repetition from baking:
 - Stir the mix 50 times
 - Knead the bread dough until it feels like your earlobe
 - Bake until golden-brown.
- Examples of repetition from mathematics:
 - Sum these values

- Find the smallest of these values
- Examples of repetition from everyday life:
 - Naively find a name in the phone book
 - Do I have a CD by Van Morrison?
- Examples of repetition from previous Scheme exercises:
 - How long is this list?
 - Is X a member of this list?

Recursion

- In Scheme, the most common mechanism for repetition is *recursion*.
- To do something that involves repeated actions, you
 - Do one action
 - Repeat the rest
 - Combine the results if necessary.
- For example, to stir your cake mix 50 times, you stir it one time and then stir it 49 more times.
- More generally, to stir a cake mix n times, you stir it one time and then $n-1$ more times.
- Similarly, to knead dough until its the right consistency, you knead it a little, check the consistency, and, if it's not the right consistency, knead it until its the right consistency.
- In the case of mathematics, to sum a list we might add the first value to the sum of the remaining values (or add the last value to the sum of the initial values).
- There are a few key aspects to recursive design:
 - You need to know when you're done (and what to do when you're done). This aspect of recursive design is called the *base case*.
 - You need to know what to do when you're not done. Here, you should do a little, try again, and then perhaps combine the results. This aspect of recursive design is called the *recursive case*.
 - You need to be sure that you're getting closer to the base case (otherwise you'll never stop).

Recursion in Scheme

- Here's the form of a typical recursive procedure:

```
(define proc
  (lambda (val)
    (if (base-case-test)
        (base-case val)
        (combine (partof val)
                 (proc (update val))))))
```

- When the value you're working with is a list and your base case is the null list, the form is somewhat simpler:

```
(define proc
  (lambda (lst)
    (if (null? lst)
        null-case
        (combine (onestep (car val))
                  (proc (cdr val))))))
```

Lab

- Start the lab on recursion.
 - Be prepared to reflect.
-

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.