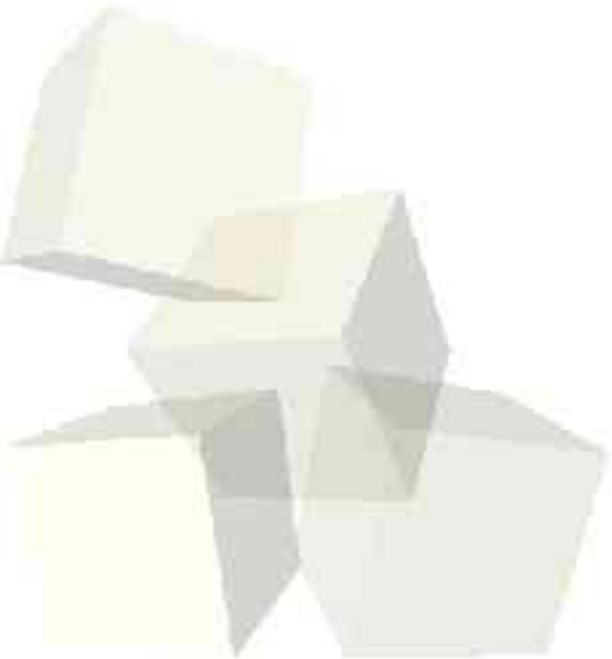
Scheme Basics

8-30-2004



Opening Discussion

- Do you have any questions about the course based on what we talked about last time?
- Naughty Dog uses a LISP derivative for a lot of their in-house work.
- If you aren't on it yet, you should register for the CS majors e-mail list. There are directions at www.cs.trinity.edu.
- Never reboot these machines unless they are completely hung. The Xena machines are single boot only so there is no reason to reboot them.

The Scheme Environment

- There are many environments that you use use to program in scheme. On these systems you can use scm or drscheme.
- scm is a simple text implementation of scheme.
- DrScheme is a more sophisticated GUI implementation.
- If you look out on the web, you can find lots of other implementations as well.

Numbers and Symbols

- Symbols/names in Scheme are case insensitive and can include numbers as well as +, -, or . as long as it starts with a character.
- Numbers are recognized as one would expect.
- Commands in Scheme are given with parenthesis around the command and arguments.

Define and Quote

- The define command allows up to binds names to things. It has the following format.
  - (define *var expression*)
- We can also tell Scheme to not evaluate something, but use it directly with the quote command.
  - (quote *symbol*)
- The quote command can be abbreviated with a single quote.
  - '*symbol*

Prefix Notation

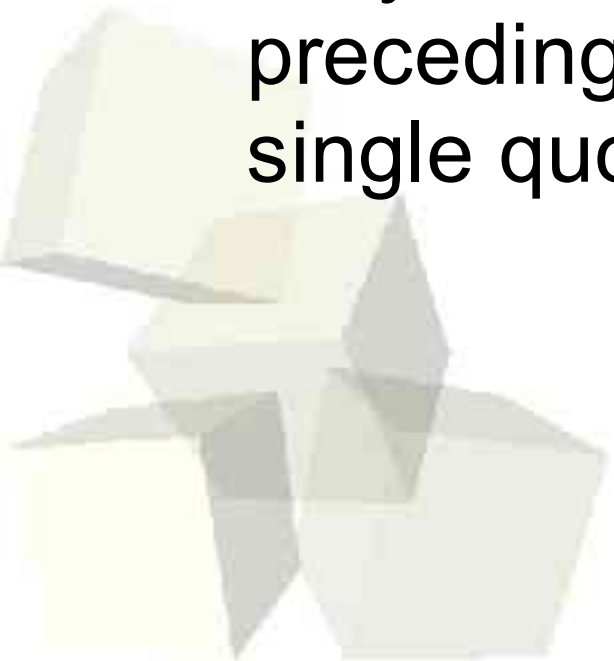
- Scheme uses prefix notation for functions and mathematical operators.
- For instance, we can add two numbers by doing something like this:
  - (+ 5 7)
- Note that this also allows us to have more than two operands.
  - (+ 3 5 7 11)
- The operands can be function calls as well.
  - (* (+ 3 5) (- 6 3))

Constructing Lists

- The most common data type in Scheme, above the atomic numbers and symbols, is the list.
- Lists are built with the cons function which appends two elements together. Technically, cons creates a memory cell with two pointers and that cells points to the two arguments passed to cons.
- For a list, the first element is what we are adding and the second is a list. We start to process adding something to '().

Quoting Lists

- For today, using cons to build all of our lists can be a bit of a pain because we can't yet build our own functions that put together lists. Thankfully lists can also be quoted.
- So you can define arbitrarily complex lists by preceding whatever list you want with a single quote.



Taking Lists Apart

- Lists are taken apart with two somewhat oddly named methods: car and cdr. These come from their relation to the memory setup on the IBM 704 where LISP was first implemented. They stand for “contents of address-register” and “contents of decrement-register”.
- car returns the first element of the list (which could be a list itself).
- cdr returns the remainder of the list.

Predicates

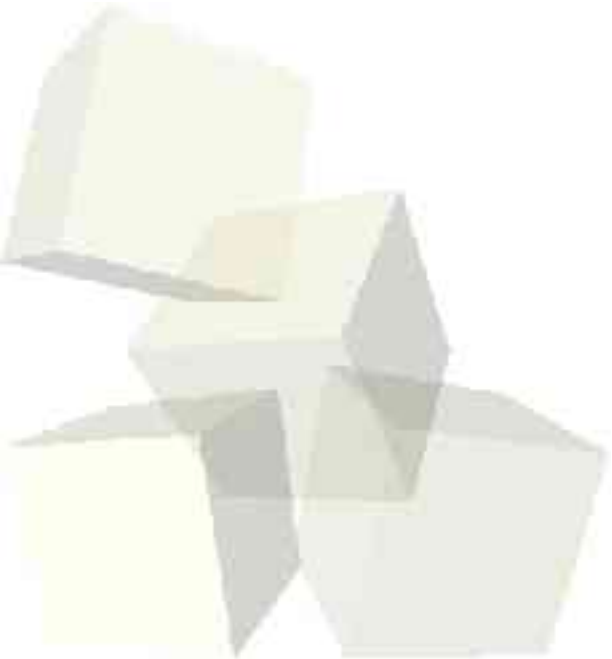
- In addition to the three basic list functions, Scheme has a number of functions called predicates that return boolean values based on their arguments.
- In Scheme, #t is true and #f or () is false.
  - number?
  - symbol?
  - boolean?
  - pair? - is it a cons cell
  - null? - is it ()
  - procedure?

Comparison Predicates

- Numbers can be compared with =, <>, >, <, >=, or <=.
- Symbols can be tested for equivalence with eq?.
- If we have two atoms items, but don't know their exact type we can use eqv? to compare them.
- To test anything, including lists, for equality we can use the equal? predicate. This is the slowest option though.

Playing with Lists

- Now that we know a little bit about lists and some basics of Scheme, we should all log int and start Scheme and play with it a little bit.



Minute Essay

- Write a line to give the list (A (B C) (D E)) the name mylist without using the quote function (use cons). Then tell me what you would get by doing the following.
  - (car mylist)
  - (car (cdr mylist))
  - (cdr (cdr mylist))
- Also remember that you can put in any feedback you want on these.