Announcements

• Homework 7 due Tuesday 11/5 @ 11:59pm.
• Project 1 composition revisions due Thursday 11/7 @ 11:59pm.
• Midterm 2 is graded.
  • (And yes, it was very challenging.)
  • Mean: 38
  • Solutions will be posted and exams distributed soon.

Scheme

Scheme is a Dialect of Lisp

What are people saying about Lisp?
- "The greatest single programming language ever designed."
  - Alan Kay, co-inventor of Smalltalk and OOP
- "The only computer language that is beautiful."
  - Neal Stephenson, DeNero’s favorite sci-fi author
- "God’s programming language."
  - Brian Harvey, Berkeley CS instructor extraordinaire

Scheme Fundamentals

Scheme programs consist of expressions, which can be:
* Primitive expressions: 2 3.3 true + quotient ...
* Combinations: (quotient 10 2) (not true) ...

Numbers are self-evaluating; symbols are bound to values.

Call expressions include an operator and 0 or more operands in parentheses.

> (quotient 10 2)
5
> (quotient (+ 8 7) 5)
3
> (+ 3 5)
8
> (+ 2 4)

"quotient" names Scheme’s built-in integer division procedure (i.e., function)

Combinations can span multiple lines (spacing doesn’t matter)

(Demo)
Special Forms

A combination that is not a call expression is a special form:

- If expression: `(if <predicate> <consequent> <alternative>)`
- And and or: `(and <e1> ... <en>)` or `(or <e1> ... <en>)`
- Binding symbols: `(define <symbol> <expression>)`
- New procedures: `(define (<symbol> <formal parameters>) <body>)`

Example: Counting Binary Trees

The structure of a sentence can be described by a tree. Each sub-tree is a constituent.

```
W    X    Y    Z
```

The number of trees over n leaves with k leaves in the left and n-k in the right is:

\[
\text{(The number of trees with } k \text{ leaves)} \times \text{(The number of trees with } n-k \text{ leaves)}
\]

(Demo)

Lambda Expressions

Lambda expressions evaluate to anonymous procedures.

```
(lambda (<formal-parameters>) <body>)
```

Two equivalent expressions:

```
(define (plus4 x) (+ x 4))
(define plus4 (lambda (x) (+ x 4)))
```

An operator can be a call expression too:

```
((lambda (x y z) (+ x y (square z))) 1 2 3)
```

Evaluates to the `add-x-y-z-square` procedure

(Demo)
Pairs and Lists

In the late 1950s, computer scientists used confusing names.

- **cons**: Two-argument procedure that creates a pair
- **car**: Procedure that returns the first element of a pair
- **cdr**: Procedure that returns the second element of a pair
- **nil**: The empty list

They also used a non-obvious notation for recursive lists.

- A (recursive) list in Scheme is a pair in which the second element is nil or a Scheme list.
- Scheme lists are written as space-separated combinations.
- A dotted list has any value for the second element of the last pair; maybe not a list!

```scheme
> (define x (cons 1 2))
> x
(1 . 2)
> (car x)
1
> (cdr x)
2
> (cons 1 (cons 2 (cons 3 (cons 4 nil))))
(1 2 3 4)
Not a well-formed list!
```

Symbolic Programming

Symbols normally refer to values; how do we refer to symbols?

```lisp
> (define a 1)
> (define b 2)
> (list a b)
(1 2)
Quotation is used to refer to symbols directly in Lisp.

> (list 'a 'b)
(a b)
> (list 'a b)
(a 2)
Quotation can also be applied to combinations to form lists.

> (car '(a b c))
(a)
> (cdr '(a b c))
(b c)
```

Scheme Lists and Quotation

Dots can be used in a quoted list to specify the second element of the final pair.

```lisp
> (cdr (cdr '(1 . 3)))
3
However, dots appear in the output only of ill-formed lists.

> '(1 2 . 3)
(1 2 . 3)
> '(1 2 (3 4))
(1 2 3 . 4) nil
> '(1 2 3 . nil)
(1 2 3)
What is the printed result of evaluating this expression?

> (cdr '((1 2) . (3 4 . (5))))
(3 4 5)
```

Sierpinski's Triangle

(Demo)