INFORMATION and REMINDERS

- Homework 5: extension until Monday, November 13, 11:59pm.
- Homework 6 will be posted by Saturday.
- Second programming project will be posted next week.
- Final exam: Wednesday, December 20, 4:00-7:00pm.

DO YOU HAVE A CONFLICT?

http://sasundergrad.rutgers.edu/forms/final-exam-conflict

- More than two (2) final exams on one calendar day
- More than two (2) final exams scheduled in consecutive periods
- Two final exams scheduled for the same exam period.
Scheme: Functions as Values (Higher-order)

Functions as arguments:

\[
\text{(define } f \text{ (lambda (g x) (g x)))}
\]

- \((f \text{ number? 0})\)
  \(\Rightarrow (\text{number? 0}) \Rightarrow \#t\)

- \((f \text{ length '(1 2)})\)
  \(\Rightarrow (\text{length '}(1 2)) \Rightarrow 2\)

- \((f \text{ (lambda (x) (* 2 x)) 3})\)
  \(\Rightarrow ((\text{lambda (x) (* 2 x)) 3})\)
  \(\Rightarrow (* 2 3) \Rightarrow 6\)

REMINDER: Computation, i.e., function application is performed by reducing the initial S-expression (program) to an S-expression that represents a value. Reduction is performed by substitution, i.e., replacing formal by actual arguments in the function body.

Examples for S-expressions that directly represent values, i.e., cannot be further reduced:

- function values (e.g.: \((\text{lambda}(x) e))\)
- constants (e.g.: 3, \#t)
Functions as returned values:

\[(\text{define } \text{plusn} \ (\text{lambda} \ (n) \ (\text{lambda} \ (x) \ (+ \ n \ x))))\]

- \((\text{plusn} \ 5)\) evaluates to a function that adds 5 to its argument

  \textit{Question}: How would you write down the value of \((\text{plusn} \ 5)\)?

- \(((\text{plusn} \ 5) \ 6) \Rightarrow 11\)
Higher-order Functions (Cont.)

In general, any n-ary function

\[(\text{lambda} \; (x_1 \; x_2 \; \ldots \; x_n) \; e)\]

can be rewritten as a nest of \(n\) unary functions:

\[(\text{lambda} \; (x_1) \text{ (lambda} \; (x_2) \text{ ( ... (lambda} \; (x_n) \; e \text{ ) ...)))}\]

This translation process is called \textit{currying}. It means that having functions with multiple parameters do not add anything to the expressiveness of the language.

\textit{Question}: How to write an application of the original vs. the curried version?

\[ (((\text{lambda} \; (x_1 \; x_2 \; \ldots \; x_n) \; e) \; v_1 \; v_2 \; \ldots \; v_n)\]

\[ ((( \ldots (((\text{lambda} \; (x_1) \; \text{ (lambda} \; (x_2) \text{ ( ... (lambda} \; (x_n) \; e \text{ ) ...))) \; v_1) \; v_2) \; \ldots \; v_n)\]
Higher-order Functions: map

(define map
  (lambda (f l)
    (if (null? l)
        ()
        (cons (f (car l)) (map f (cdr l)))
    ))
)

• map takes two arguments: a function and a list
• map builds a new list by applying the function to every element of the (old) list
Higher-order Functions: map

- Example:
  \[(\text{map abs } '(-1 2 -3 4)) \Rightarrow (1 2 3 4)\]
  \[(\text{map (lambda (x) (+ 1 x)) } '(-1 2 -3)) \Rightarrow (0 3 -2)\]

- Actually, the built-in map can take more than two arguments:
  \[(\text{map + } '(1 2 3) '(4 5 6)) \Rightarrow (5 7 9)\]
More on Higher Order Functions

**reduce**

Higher order function that takes a binary, associative operation and uses it to “roll-up” a list

\[
\text{(define reduce}
\text{  (lambda (op 1 id)}
\text{    (if (null? 1)}
\text{      id}
\text{      (op (car 1) (reduce op (cdr 1) id)) )))}
\]

Example:

\[
\text{(reduce + ’(10 20 30) 0) ⇒}
\text{( + 10 (reduce + ’(20 30) 0)) ⇒}
\text{( + 10 (+ 20 (reduce + ’(30) 0))) ⇒}
\text{( + 10 (+ 20 (+ 30 (reduce + ’() 0)))) ⇒}
\text{( + 10 (+ 20 (+ 30 0))) ⇒}
\text{60}
\]
More on Higher Order Functions

Now we can compose higher order functions to form compact powerful functions

Examples:

```
(define sum
  (lambda (f l)
    (reduce + (map f l) 0))

(sum (lambda (x) (* 2 x)) '(1 2 3)) ⇒

(reduce (lambda (x y) (+ 1 y)) '(a b c) 0) ⇒
```
Next Lecture

Next time:

- More programming in Scheme
- Lambda calculus