More Lambda

CSE 413, Autumn 2002
Programming Languages

Readings and References

- Reading
  » Section 1.3, *Structure and Interpretation of Computer Programs*, by Abelson, Sussman, and Sussman, but you've already read this, right?

- Other References

Procedures as unnamed blobs

- With \texttt{lambda}, we've separated the body of the procedure from any particular name for the procedure
- Procedures are objects like any other, and can be handed around from procedure to procedure as arguments, return values, etc
- Procedures can be defined and applied without ever getting a name assigned to them

A numeric interval

$x = [0:.1:1];$
$y = \sin(x);$  
plot(x,y)
calculate-h

; define a function to calculate an interval size \((b-a)/n\)

\[
\text{(define calculate-h (lambda (a b n) (/ (- b a) n)))}
\]

... and bind it to the name calculate-h

; try it out on \([0,1]\)
\[
\text{(calculate-h 0 1 10)}
\]

apply the function to some arguments

anonymous calculate-h

; do the same thing without naming the function

\[
\text{((lambda (a b n) (/ (- b a) n)) 0 1 10)}
\]

... and apply it to some arguments

calculate last-x

; define a function that figures out what the beginning of the last interval is

; calculate \(a+(h*(n-1))\) directly

\[
\text{(define (last-x1 a b n)}
\[
\text{(+ a (* (- n 1) (/ (- b a) n))))}
\]

last-x using a helper function

; calculate \(a+(k*h)\) using a simple function, and pre-calculate \(k\) and \(h\) to pass to the function

\[
\text{(define (last-x2 a b n)}
\[
\text{(+ a (* k h)))}
\]
\[
\text{(define (use-kh k h)}
\[
\text{(+ a (* k h)))}
\]
\[
\text{(use-kh (- n 1) (/ (- b a) n))}
\]

define a function body and bind it to the name use-kh

apply use-kh to some arguments
**last-x using anonymous helper function**

; calculate \( a + (k \cdot h) \) using an anonymous function

\[
\text{define (last-x3 \ a \ b \ n)} \\
((\text{lamba} \ (k \ h) \ (+ \ a \ (* \ k \ h)))) \\
(- \ n \ 1) \\
(/ \ (- \ b \ a) \ n))
\]

... and apply it to some arguments

**last-x with concealed anonymous function**

; hide the use of the anonymous function by using \texttt{let}

\[
\text{define (last-x4 \ a \ b \ n)} \\
(\text{let} \ ((h \ (/ \ (- \ b \ a) \ n)) \ \\
\quad (k \ (- \ n \ 1)) \ \\
\quad (+ \ a \ (* \ k \ h))))
\]

... and use those names in the body of the \texttt{let}

---

**Special form \texttt{let}**

\[
\text{(let} \ ((\langle \text{var} \rangle \ \langle \text{exp} \rangle)) \\
\quad ((\langle \text{var} \rangle \ \langle \text{exp} \rangle)) \\
\quad \langle \text{body} \rangle)
\]

- When the \texttt{let} is evaluated, each expression \texttt{exp}_i is evaluated and the resulting value is associated with the related name \texttt{var}_i, then the \texttt{body} is evaluated.
- There is no order implied in the evaluation of \texttt{exp}_i
- This is exactly the same as parameter evaluation before a procedure call
  - This is parameter evaluation before a procedure call

**scope and \texttt{let}**

; an example in scoping with \texttt{let}

\[
\text{define x 2} \\
(\text{let} \ ((x \ 3) \ \\
\quad (y \ (+ \ x \ 2))) \ \\
\quad (* \ x \ y))
\]

\[
((\text{lamba} \ x \ y) \\
\quad (* \ x \ y)) \\
\quad 3 \\
\quad (+ \ x \ 2))
\]

the parameter values are calculated outside the scope of the parameter variables in the procedure

this \texttt{let} and this lambda are equivalent

scope of the local x and y

the parameter values are calculated outside the scope of the parameter variables in the procedure
nesting lets lets us get x

; nested lets and let*
(define x 2)

(let ((x 3))
  (let ((y (+ x 2)))
    (* x y)))

(let* ((x 3)
        (y (+ x 2)))
  (* x y))

is this x referenced anywhere?

Special form let*

(let* ((\(\langle\text{var}_1\rangle\ \langle\text{exp}_1\rangle\))
        ((\langle\text{var}_2\rangle\ \langle\text{exp}_2\rangle))))

\(\langle\text{body}\rangle\)

- When the let* is evaluated, each expression \(\text{exp}_i\) is evaluated in turn and the resulting value is associated with the related name \(\text{var}_i\), then the body is evaluated.
- The \(\text{exp}_i\) are evaluated in left to right order
  » each binding indicated by \((\langle\text{var}_i\rangle\ \langle\text{exp}_i\rangle)\) is part of the environment for \((\langle\text{var}_{i+1}\rangle\ \langle\text{exp}_{i+1}\rangle)\) and following
  » This is exactly equivalent to nesting the let statements

an iterator with parameter h

; show all the x values on the interval
(define (show-x1 a b n)
  (define (iter h count)
    (if (> count n)
      (newline)
      (begin
        (display (+ a (* h count)))
        (display " ")
        (iter (+ count 1))))
  (iter (/ (- b a) n) 0))

h defined in enclosing scope

; show all the x values on the interval
; using let
(define (show-x2 a b n)
  (let ((h (/ (- b a) n)))
    (define (iter count)
      (if (> count n)
        (newline)
        (begin
          (display (+ a (* h count)))
          (display " ")
          (iter (+ count 1))))
    (iter 0)))
**Special form begin**

`(begin <exp1> <exp2> ... <expn>)`

- Evaluate the `expi` in sequence from left to right
- The value returned by the entire `begin` expression is the value of `expn`
- Best used to sequence side effects like I/O
  - for example displaying each of the x values in `show-x`
- There is implicit sequencing in the body of a `lambda` procedure or a `let` but we generally don't use it
  - the procedure returns the value of the last `expi`, so the body of most of our procedures consists of one expression only

```
; show all the x values on the interval
(define (show-x2 a b n)
  (let ((h (/ (- b a) n)))
    (define (iter count)
      (if (> count n)
       (newline)
       (begin
         (display (+ a (* h count)))
         (display " ")
         (iter (+ count 1))))
    (iter 0)))
```

---

**sequencing with begin**

Welcome to DrScheme, version 201.
Language: Standard (R5RS).
> (show-x2 0 1 10)
0 1/10 1/5 3/10 2/5 1/2 3/5 7/10 4/5 9/10 1
>