6.001 recitation 18         4/25/07

- interpretation
- our evaluator

Dr. Kimberle Koile
interpretation  key ideas

- abstractions
- cycle
- environment
- parts
\textit{stages of an interpreter}

\texttt{"(average 40 (+ 5 5))"}

25

\texttt{"25"}
eval: dispatch on expression type

apply: eval args then apply operator
our evaluator

; the initial global environment
(define GE
  (extend-env-with-new-frame
   (list 'plus* 'greater*)
   (list (make-primitive +) (make-primitive >))
   '()))

(define (eval exp env)
  (cond
   ((number? exp) exp)
   ((symbol? exp) (lookup exp env))
   ((define? exp) (eval-define exp env))
   ((if? exp) (eval-if exp env))
   ((lambda? exp) (eval-lambda exp env))
   ((let? exp) (eval-let exp env))
   ((application? exp)
     (apply* (eval (car exp) env)
     (map (lambda (e) (eval e env))
       (cdr exp))))
   (else
    (error "unknown expression " exp))))
(define (eval exp env)
  (cond
    ...
...
    
    ((if? exp) (eval-if exp env))
    ...
  ))

Example
exp: (if (= n o)
    'done
    (do-something n))

our evaluator
our evaluator

Example

define (if exp) (tag-check exp 'if*)

(define (eval exp env)
  (cond ...

  ((if? exp) (eval-if exp env)) ...
))

(define (eval exp env)
  (cond ...

  ((if? exp) (eval-if exp env)) ...
))
when

e.g. (when (= x 0) (print "zero"))

semantics is same as if without alternate clause
1. \texttt{(quote* expr)} returns \texttt{expr} without evaluating it. Assume \texttt{eval} calls \texttt{eval-quote} if the procedure \texttt{quote?} is true for a given \texttt{quote*} statement. Write \texttt{eval-quote}, which takes one argument.
2. (eval-sequence exps env) evaluates each expression in a list of expressions, and returns the value of the last one. Assume eval calls eval-sequence if the procedure sequence? is true for a given expression. Write eval-sequence, which takes two arguments, expr and env.

(Hint: You’ll need to call begin.)
3. \textbf{(case*) expr}
   \texttt{((val1 val2 \ldots) consequent)}
   \texttt{((vali valj \ldots) consequent)}
   \ldots
   \texttt{(else* alternative))}

Case* evaluates expr and compares its value (using eqv?) against each of the listed values, which are not evaluated. When a match is found, the corresponding consequent expression is evaluated and returned as the result of the case*. If no matches are found, the alternate expression is evaluated and returned instead. You can assume the else* clause is required if you like.

Assume eval calls eval-case if the procedure case? is true for a given case* statement.

\textbf{(define (eval-case exp env)}
\texttt{  (let ((target-value (eval (second exp) env)))}
\texttt{    (eval-case-clauses target-value (cddr exp) env)))}

On the next slide, write \textbf{eval-case-clauses}, which takes three arguments: a target-value, a list of clauses, and env.
Write eval-case-clauses.
4. \((\textbf{begin}* \text{ expr1 expr2 \ldots exprn})\) evaluates each expression in the sequence, returning the value of \text{exprn} as its final result. Assume \text{eval} calls \text{eval-begin} if the procedure \text{begin?} returns true for a given \text{begin*} statement.

\begin{verbatim}
(define (begin? exp)  (tag-check exp 'begin*))

(define (eval-begin exp env)
  (eval-begin-body (cdr exp) env))
\end{verbatim}

Write \text{eval-begin-body}, which takes two arguments, body and env.