MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall 2007

Recitation 18 — 11/7/2007 Solutions
Interpretation

Interpreter code from lecture, with only a few minor modifications. This version still uses almost no abstractions.

```
(define (eval exp env)
  (cond ((number? exp) exp)
        ((symbol? exp) (lookup exp env))
        ((eq? (car exp) 'quote) (cadr exp))
        ((eq? (car exp) 'cond)
         (eval-clauses (cdr exp) env))
        ((eq? (car exp) 'lambda)
         (eval-lambda exp env))
        (else
         (apply (eval (car exp) env)
                (values (cdr exp) env)))))

(define (apply proc args)
  (cond ((primitive? proc) (papply proc args))
        ((eq? (car proc) 'procedure)
         (let ((parameters (cadr proc))
               (body (caddr proc))
               (env (cadddr proc)))
           (eval body
                 (extend env
                         parameters
                         args))))
        (else (error "not a proc"))))

(define (eval-lambda exp env)
  (let ((parameters (cadr exp))
        (body (caddr exp)))
    (list 'procedure parameters body env)))

(define (values exps env)
  (cond ((null? exps) '())
        (else (cons (eval (car exps) env)
                    (values (cdr exps) env)))))

(define (eval-clauses clauses env)
  (cond ((null? clauses) '())
        ((eq? (caar clauses) 'else)
         (eval (cadar clauses) env))
        ((not (eq? (eval (caar clauses) env) #f))
         (eval (cadar clauses) env))
        (else (eval-clauses (cdr clauses) env))))
```

```
(define (extend base-env vars values)
  (cons (make-frame vars values) base-env))

(define (make-frame vars vals)
  (cond ((null? vars) (cond ((null? vals) '())
                            (else (error "too many args"))))
        ((null? vals) (error "too few args"))
        (else (cons (cons (car vars) (car vals))
                    (make-frame (cdr vars) (cdr vals))))))

(define (lookup var env)
  (cond ((null? env) (error "unbound variable" var))
        (else (let ((binding (assq var (car env))))
                (cond (binding (cdr binding))
                      (else (lookup var (cdr env)))))))))

(define (assq var bindings)
  (cond ((null? bindings) #f)
        ((eq? (caar bindings) var) (car bindings))
        (else (assq var (cdr bindings)))))

(define (primitive? proc)
  (memq proc (list car cdr cons eq? + = > <)))

(define (papply proc args)
  (cond ((eq? proc car) (car (car args)))
        ((eq? proc cdr) (cdr (car args)))
        ((eq? proc cons) (cons (car args) (cadr args)))
        ((eq? proc eq?) (eq? (car args)))
        ((eq? proc +) (+ (car args) (cadr args)))
        ((eq? proc >) (> (car args) (cadr args)))
        ))
```

Problems

Suppose the global environment is defined as such:

```
(define *GE*
  (extend '()
          '(car cdr cons eq? + >)
          (list car cdr cons eq? + >)))
```

Evaluate the following, in order

1. `(eval 'x *GE*)`

   error "unbound variable" x

2. `(eval '(eval '(> x 4)`
   `(extend *GE*`

```
                '(x)
                (list 3)))
```

#f

3. `(eval '((lambda (x) (+ x 2)) 5) *GE*)`

   7

4. ```
   (eval '(cond ((> x 4) (quote a))
                ((> 1 x) (quote b))
                (else (quote c)))
         (extend *GE*
                 '(x)
                 (list 2)))
   ```

   c

5. Add a new special form: `begin`

   ```
   (define (eval-begin exp env)
     (eval-seqnence (cdr exp) env)

   (define (eval-sequence exp env)
     (cond ((null? exp) '())
           ((null? (cdr exp))
            (eval (car exp) env))
           (else
            (eval (car exp) env)
            (eval-sequence (cdr exp) env))))
   ```
   ```

6. What other standard special forms are missing? How would you add them?