MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Spring 2006

Recitation 21 — 5/3/2006
Interpretation

Interpreter code from lecture: (only slightly modified)

```
(define (star-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))
         ((application? exp) (star-apply (star-eval (car exp) env)
                                         (map (lambda (e) (star-eval e env))
                                              (cdr exp))))
   (else (error "unknown expression" exp))))

(define (star-apply operator operands)
  (cond ((primitive? operator)
         (apply (get-scheme-procedure operator) operands))
        ((compound? operator)
         (star-eval (body operator)
                    (extend-env-with-new-frame
                     (parameters operator)
                     operands
                     (env operator))))
         (else (error "operator not a procedure" operator))))

(define (eval-define exp env)
  (let ((name (cadr exp))
        (defined-to-be (caddr exp)))
    (table-put! (car env) name (star-eval defined-to-be env))
    'undefined))

(define (eval-if exp env)
  (let ((predicate (cadr exp))
        (consequent (caddr exp))
        (alternative (cadddr exp)))
    (let ((test (star-eval predicate env)))
      (cond
       ((eq? test #t) (star-eval consequent env))
       ((eq? test #f) (star-eval alternative env))
       (else (error "predicate not boolean"
                    predicate))))))

(define (eval-lambda exp env)
  (let ((args (cadr exp))
        (body (caddr exp)))
    (make-compound args body env)))
```

The environment model:

```
(define (extend-env-with-new-frame names values env)
  (let ((new-frame (make-table)))
    (make-bindings! names values new-frame)
    (cons new-frame env)))

(define (make-bindings! names values table)
  (for-each
   (lambda (name value) (table-put! table name value))
   names values))

(define (lookup name env)
  (if (null? env)
      (error "unbound variable" name)
      (let ((binding (table-get (car env) name)))
        (if binding
            (binding-value binding)
            (lookup name (cdr env))))))
```

Other Helper Procedures:

```
(define prim-tag 'primitive)
(define (make-primitive scheme-proc) (list prim-tag scheme-proc))
(define (primitive? e) (tag-check e prim-tag))
(define (get-scheme-procedure prim) (cadr prim))

(define compound-tag 'compound)
(define (make-compound parameters body env)
  (list compound-tag parameters body env))
(define (compound? exp) (tag-check exp compound-tag))
(define (parameters compound) (cadr compound))
(define (body compound) (caddr compound))
(define (env compound) (cadddr compound))

(define (binding-value binding) (cadr binding))

(define (tag-check e sym)
  (and (pair? e) (eq? (car e) sym)))

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

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

(define (lambda? exp)
  (tag-check exp 'lambda*))

(define (application? exp)
  (list? exp))
```

```
;;;table definition

(define table-tag 'table)

;;; table -> data
(define get-table-data cadr)

;;; table, data -> undef
(define set-table-data!
  (lambda (table new-data)
    (set-car! (cdr table) new-data)))

;;; void -> table
(define make-table
  (lambda () (list table-tag '())))

;;; table, symbol -> (binding | null)
(define table-get
  (lambda (table name)
    (assq name (get-table-data table))))

;;; table, symbol, anytype -> undef
(define table-put!
  (lambda (table name value)
    (set-table-data! table (cons (list name value)
                                 (get-table-data table)))))
```

Problems

Suppose the global environment is defined as such:

```
(define GE
  (extend-env-with-new-frame
   (list 'plus* 'greater* 'true*)
   (list (make-primitive +) (make-primitive >) #t)
   '()))
```

Evaluate the following, in order

1. `(star-eval 'x GE)`

2. `(star-eval '(define* x 6) GE)`

3. `(star-eval '(define* foo (lambda* (y) (plus x y))) GE)`

4. `(star-eval '(foo 4) GE)`

5. Add a new special form: `begin*`

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

6. Add another new special form: `cond*`

   ```
   (define (eval-cond exp env)
   ```