

1. Arithmetic calculator

```
1
2
3 (define (tag-check e sym) (and (pair? e) (eq? (car e) sym)))
4 (define (sum? e) (tag-check e 'plus*))
5
6 (define (eval exp)
7   (cond
8     ((number? exp) exp)
9     ((sum? exp) (eval-sum exp))
10    (else
11     (error "unknown expression " exp))))
12
13 (define (eval-sum exp)
14   (+ (eval (cadr exp)) (eval (caddr exp))))
15
16
17 (eval '(plus* 24 (plus* 5 6)))
18
```

2. Names

```
1
2
3 (define (define? exp) (tag-check exp 'define*))
4
5 (define (eval exp)
6   (cond
7     ((number? exp) exp)
8     ((sum? exp) (eval-sum exp))
9     ((symbol? exp) (lookup exp))
10    ((define? exp) (eval-define exp))
11    (else
12     (error "unknown expression " exp))))
13
14 ; variation on table ADT from March 2 lecture (only difference is
15 ; that table-get returns a binding, while original version
16 ; returned a value):
17 ; make-table      void -> table
18 ; table-get       table, symbol -> (binding | null)
19 ; table-put!      table, symbol, anytype -> undef
20 ; binding-value   binding -> anytype
21
22 (define environment (make-table))
23
24 (define (lookup name)
25   (let ((binding (table-get environment name)))
26     (if (null? binding)
27         (error "unbound variable: " name)
28         (binding-value binding))))
29
30 (define (eval-define exp)
31   (let ((name (cadr exp))
32         (defined-to-be (caddr exp)))
33     (table-put! environment name (eval defined-to-be))
34     'undefined))
35
36 (eval '(define* x* (plus* 4 5)))
37 (eval '(plus* x* 2))
38
39
40
41 ; Index to procedures that have not changed:
42 ; procedure      page      line
43 ; sum?           1         4
44 ; eval-sum      1         13
45
```

3. Conditionals and if

```
1
2
3 (define (greater? exp) (tag-check exp 'greater*))
4 (define (if? exp)      (tag-check exp 'if*))
5
6 (define (eval exp)
7   (cond
8     ((number? exp) exp)
9     ((sum? exp)    (eval-sum exp))
10    ((symbol? exp) (lookup exp))
11    ((define? exp) (eval-define exp))
12    ((greater? exp) (eval-greater exp))
13    ((if? exp)      (eval-if exp))
14    (else
15     (error "unknown expression " exp))))
16
17 (define (eval-greater exp)
18   (> (eval (cadr exp)) (eval (caddr exp))))
19
20 (define (eval-if exp)
21   (let ((predicate (cadr exp))
22         (consequent (caddr exp))
23         (alternative (caddr exp)))
24     (let ((test (eval predicate)))
25       (cond
26         ((eq? test #t) (eval consequent))
27         ((eq? test #f) (eval alternative))
28         (else          (error "predicate not a conditional: "
29                               predicate))))))
30
31 (eval '(define* y* 9))
32 (eval '(if* (greater* y* 6) (plus* y* 2) 15))
33
34
35 ; Index to procedures that have not changed:
36 ;   procedure           page      line
37 ;   sum?                 1         4
38 ;   eval-sum             1        13
39 ;   lookup                2        22
40 ;   define?              2         3
41 ;   eval-define          2        28
42
43
```

4. Store operators in the environment

```
1
2
3 (define (application? e) (pair? e))
4
5 (define (eval exp)
6   (cond
7     ((number? exp)      exp)
8     ((symbol? exp)      (lookup exp))
9     ((define? exp)      (eval-define exp))
10    ((if? exp)           (eval-if exp))
11    ((application? exp) (apply (eval (car exp))
12                                (map eval (cdr exp))))
13    (else
14     (error "unknown expression " exp))))
15
16 ;; rename scheme's apply so we can reuse the name
17 (define scheme-apply apply)
18
19 (define (apply operator operands)
20   (if (primitive? operator)
21       (scheme-apply (get-scheme-procedure operator) operands)
22       (error "operator not a procedure: " operator)))
23
24 ;; primitive: an ADT that stores scheme procedures
25
26 (define prim-tag 'primitive)
27 (define (make-primitive scheme-proc)(list prim-tag scheme-proc))
28 (define (primitive? e) (tag-check e prim-tag))
29 (define (get-scheme-procedure prim) (cadr prim))
30
31 (define environment (make-table))
32 (table-put! environment 'plus* (make-primitive +))
33 (table-put! environment 'greater* (make-primitive >))
34 (table-put! environment 'true* #t)
35
36 (eval '(define* z* 9))
37 (eval '(plus* 9 6))
38 (eval '(if* true* 10 15))
39
40
41 ; Index to procedures that have not changed:
42 ;   procedure      evaluator   line
43 ;   lookup         2           22
44 ;   define?        2           3
45 ;   eval-define    2           28
46 ;   if?            3           4
47 ;   eval-if        3           20
```

5. Environment as explicit parameter

;This change is boring! Exactly the same functionality as #4.

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

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

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

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

(eval '(define* z* (plus* 4 5)) environment)
(eval '(if* (greater* z* 6) 10 15) environment)
```

Index to procedures that have not changed:

procedure	evaluator	line
define?	2	3
if?	3	4
application?	4	3
apply	4	19

50
51

6. Defining new procedures

```
1
2
3 (define (lambda? e) (tag-check e 'lambda*))
4
5 (define (eval exp env)
6   (cond
7     ((number? exp)      exp)
8     ((symbol? exp)      (lookup exp env))
9     ((define? exp)      (eval-define exp env))
10    ((if? exp)          (eval-if exp env))
11    ((lambda? exp)       (eval-lambda exp env))
12    ((application? exp) (apply (eval (car exp) env)
13                                (map (lambda (e) (eval e env))
14                                     (cdr exp)))))
15   (else
16     (error "unknown expression " exp))))
17
18 (define (eval-lambda exp env)
19   (let ((args (cadr exp))
20         (body (caddr exp)))
21     (make-compound args body env)))
22
23 (define (apply operator operands)
24   (cond ((primitive? operator)
25         (scheme-apply (get-scheme-procedure operator)
26                       operands))
27         ((compound? operator)
28          (eval (body operator)
29                (extend-env-with-new-frame
30                  (parameters operator)
31                  operands
32                  (env operator)))))
33   (else
34     (error "operator not a procedure: " operator))))
35
36
37
38 ;; ADT that implements the "double bubble"
39
40 (define compound-tag 'compound)
41 (define (make-compound parameters body env)
42   (list compound-tag parameters body env))
43 (define (compound? exp) (tag-check exp compound-tag))
44
45 (define (parameters compound) (cadr compound))
46 (define (body compound)      (caddr compound))
47 (define (env compound)       (caddr compound))
48
49
```

1 ; Environment model code (part of eval 6)

```
2
3 ; Environment = list<table>
4
5 (define (extend-env-with-new-frame names values env)
6   (let ((new-frame (make-table)))
7     (make-bindings! names values new-frame)
8     (cons new-frame env)))
9
10 (define (make-bindings! names values table)
11   (for-each
12     (lambda (name value) (table-put! table name value))
13     names values))
14
15 ; the initial global environment
16 (define GE
17   (extend-env-with-new-frame
18     (list 'plus* 'greater*)
19     (list (make-primitive +) (make-primitive >))
20     nil))
21
22
23 ; lookup searches the list of frames for the first match
24 (define (lookup name env)
25   (if (null? env)
26       (error "unbound variable: " name)
27       (let ((binding (table-get (car env) name)))
28         (if (null? binding)
29             (lookup name (cdr env))
30             (binding-value binding)))))
31
32 ; define changes the first frame in the environment
33 (define (eval-define exp env)
34   (let ((name (cadr exp))
35         (defined-to-be (caddr exp)))
36     (table-put! (car env) name (eval defined-to-be env))
37     'undefined))
38
39
40 (eval '(define* twice* (lambda* (x*) (plus* x* x*))) GE)
41 (eval '(twice* 4) GE)
42
43 Index to procedures that have not changed:
44   procedure          evaluator   line
45   define?           2           3
46   if?               3           4
47   application?     4           3
48   eval-i
```