```
 1: #!/afs/cats.ucsc.edu/courses/cmps112-wm/usr/racket/bin/mzscheme -qr
 2: ;; $Id: hashexample.scm,v 1.2 2014-10-31 17:35:08-07 - - $
 3: ;;
 4: ;; Another hash table example, showing insertion of variables,
 5: ;; vectors, and functions, and checking for lookup.
 6: ;; Note the script above uses -qr instead of -qC.
 7: ;;
 8:
 9: ;;
10: ;; A little utility function to make printing easier.
11: ;; Mz Scheme does have a printf function, but we'll stick to
12: ;; standard Scheme here.
13: ;;
14: (define (show label it)
15:     (display label)
16:     (display " = ")
17:     (display it)
18:     (newline)
19: )
20:
21: ;;
22: ;; Create a hash table and put in some functions and variables.
23: ;;
24: (define ht (make-hash))
25: (for-each
26:     (lambda (item) (hash-set! ht (car item) (cadr item)))
27:     `((var 34)
28:       (+ ,(lambda (x y) (+ x y)))
29:       (- ,(lambda (x y) (- x y)))
30:       (* ,*)
31:       (vec ,(make-vector 10 0.0)))
32: )
33:
34: ;;
35: ;; Print the hash table.
36: ;;
37: (hash-for-each ht (lambda (key value) (show key value)))
38: (newline)
39:
40: ;;
41: ;; show the value of a simple variable.
42: ;; the last argument #f causes hash-ref to return it
43: ;; rather than crashing the program on failure to find.
44: ;;
45: (show "var" (hash-ref ht 'var #f))
46:
47: ;;
48: ;; Set a vector element, print it, and the whole vector.
49: ;;
50: (vector-set! (hash-ref ht 'vec #f) 5 3.1415926535)
51: (show "vec[5]" (vector-ref (hash-ref ht 'vec) 5))
52: (show "vec" (hash-ref ht 'vec #f))
53:
54: ;;
55: ;; A couple more examples.
56: ;;
57: (show "(+ 3 4)" (apply (hash-ref ht '+ #f) '(3 4)))
58: (show "not found" (hash-ref ht 'foo #f))
```

```
59:
60: ;;
61: ;; The function evalexpr outlines how to evaluate a list
62: ;; recursively.
63: ;;
64: (define (evalexpr expr)
65:    (cond ((number? expr) expr)
66:          ((symbol? expr) (hash-ref ht expr #f))
67:          ((pair? expr)   (apply (hash-ref ht (car expr))
68:                                 (map evalexpr (cdr expr))))
69:          (else #f))
70: )
71:
72: ;;
73: ;; Now print out the value of several expressions.
74: ;;
75: (for-each
76:     (lambda (expr) (show expr (evalexpr expr)))
77:     '( (* var 7)
78:        (- 3 4)
79:        (+ (* var 7) (- 3 4))
80: ))
81:
82: ;;
83: ;; Just to verify that we got all the way.
84: ;;
85: (display "DONE.") (newline)
```

```
 1: * = #<procedure:*>
 2: - = #<procedure:.../hashexample.scm:29:10>
 3: + = #<procedure:.../hashexample.scm:28:10>
 4: vec = #(0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0)
 5: var = 34
 6:
 7: var = 34
 8: vec[5] = 3.1415926535
 9: vec = #(0.0 0.0 0.0 0.0 0.0 3.1415926535 0.0 0.0 0.0 0.0)
10: (+ 3 4) = 7
11: not found = #f
12: (* var 7) = 238
13: (- 3 4) = -1
14: (+ (* var 7) (- 3 4)) = 237
15: DONE.
```