CS 152 / SE 152
Programming Language Paradigms

Spring Semester 2014

Department of Computer Science
San Jose State University
Prof. Ron Mak

Assignment #6

Assigned: Monday, April 14
Due: Friday, May 2 at 11:59 pm
Team assignment, 100 points max

Scheme interpreter

Use Java to write a Scheme interpreter that can execute a limited set of programs. Expand upon your work from Assignment #5.

Process special forms such as **`define`**, **`lambda`**, **`let`**, **`let*`**. Create a separate symbol table for each scope. Link the symbols tables into the parse tree as shown in class. Maintain a symbol table stack during parsing. Ensure that symbols are declared before they're used.

Maintain a runtime stack and a runtime display. Evaluate symbols and list expressions. Don't evaluate quoted symbols and lists. Handle procedure calls, parameter passing, and return values.

Print values returned to the top level.

What to turn in

Each team turns in one assignment consisting of

- The Java source files of your Scheme interpreter.
- A text file (or several) containing output from your interpreter.
- A short report (no more than one page) describing your software design and discussing any issues.

Email to: ron.mak@sjsu.edu. Your subject line must say

CS 152 Assignment #6 ***team name***

Example: **`CS 152 Assignment #6 Super Programmers`**

CC all the team members so that I can do a "Reply all" when I send out your score. This is a team assignment. Each member of the team will receive the same score.

Input files

Your interpreter must correctly evaluate the following. All can be read in from a single or multiple text files. You may assume that the input files do not contain any syntax errors.

```
(define member?
  (lambda (item lst)
    (cond
      ((null? lst) #f)
      ((equal? item (car lst)) #t)
      (else (member? item (cdr lst)))
)))

(member? 3 '(1 2 3)) ➔ #t
(member? 'b '(a (b c) d)) ➔ #f
```

```
(define remove-last
  (lambda (item lst)
    (cond
      ((null? lst) '())
      ((and (equal? item (car lst)) (not (member? item (cdr lst)))) (cdr lst))
      (else (cons (car lst) (remove-last item (cdr lst)))))
))

(remove-last 'a '(b a n a n a s)) ➔ (b a n a n s)
(remove-last '(a b) '(a b (a b) a b (b a) a b (a b) a b))
    ➔ (a b (a b) a b (b a) a b a b)
```

```
(define same-structure?
  (lambda (x y)
    (cond
      ((and (null? x) (null? y)) #t)
      ((null? x) #f)
      ((null? y) #f)
      ((and (pair? (car x)) (pair? (car y)))
       (and (same-structure? (car x) (car y))
            (same-structure? (cdr x) (cdr y))))
      (else (and (same-type? (car x) (car y))
                 (same-structure? (cdr x) (cdr y)))))
))

(define float?
  (lambda (x)
    (and (real? x) (not (integer? x)))
))

(define same-type?
  (lambda (x y)
    (or (and (symbol? x) (symbol? y))
        (and (integer? x) (integer? y))
        (and (float? x) (float? y))
        (and (boolean? x) (boolean? y))
        (and (char? x) (char? y))
        (and (string? x) (string? y)))
))

(same-structure? '(1 (a (b 3.14) ((c)))) '(3 (z (x 1.23) ((q))))) ➔ #t
(same-structure? '(1 (a (b 3.14) ((c)))) '(3 (z (x 3) ((q))))) ➔ #f
(same-structure? '(1 2 3 4 5) '(5 4 3 2)) ➔ #f
(same-structure? '() '()) ➔ #t
(same-structure? '(("hello") "world") '(("good-bye") "sam")) ➔ #t
```

```
(define sandwich-first
  (lambda (a b lst)
    (cond
      ((null? lst) '())
      ((null? (cdr lst)) lst)
      ((and (equal? b (car lst)) (equal? b (cadr lst)))
       (append (list b a b) (cddr lst)))
      (else (cons (car lst) (sandwich-first a b (cdr lst)))))
))

(sandwich-first 'meat 'bread '(bread bread)) ➔ (bread meat bread)
(sandwich-first 'meat 'bread '()) ➔ ()
(sandwich-first 'meat 'bread '(meat meat)) ➔ (meat meat)
```

```
(define min-to-head
  (lambda (lst)
    (cond
      ((null? lst) '())
      ((null? (cdr lst)) lst)
      (else (let* ((new-lst (min-to-head (cdr lst)))
                   (second (car new-lst)))
              (if (> (car lst) second)
                  (cons second (cons (car lst) (cdr new-lst)))
                  lst)))
)))

(min-to-head '(5 8 1 0 6 2 1 9)) ➔ (0 5 8 1 6 2 1 9)

(define sort
  (lambda (lst)
    (if (null? lst)
        '()
        (let ((mth (min-to-head lst)))
          (cons (car mth) (sort (cdr mth)))))
))

(sort '(5 8 1 0 6 2 1 9)) ➔ (0 1 1 2 5 6 8 9)
```

Can you catch the multiply defined symbol **a** at compile time?

```
(define mistake
  (lambda (a b a)
    (let ((d (+ a b)))
      d)))
```

Can you catch the undefined symbol **c** at compile time?

```
(define error
  (lambda (a b)
    (let ((d (+ a b c)))
      d
)))
```

Can you catch the incorrect number of parameters at run time?

```
(member? 'a)
```

Extra Credit!

Run the suggested solution to Assignment #3 in your Scheme interpreter.
http://www.cs.sjsu.edu/~mak/CS152/assignments/3/Assignment3Solution.ss

```
(define f '(a x ^ 5 + b x ^ 4 + 2 x ^ 3 + 6 x ^ 2 + 3 x + 7))
(deriv f 'x) ➔ (5 a x ^ 4 + 4 b x ^ 3 + 6 x ^ 2 + 12 x + 3)

(define a 1)
(define b 1)

(evaluate f 0) ➔ 7
(evaluate f 1) ➔ 20

(evaluate-deriv f 0) ➔ 3
(evaluate-deriv f 1) ➔ 30
```

Up to 50 Assignment points for a team or individual student.
Extra credit due Friday, May 9.