TEL AVIV UNIVERSITY
Department of Computer Science
0368.1105 – Extended Introduction to Computer Science
Fall Semester, 2000/2001

Homework 10, Jan 25, 2001

This homework is due by 18:00 on Thursday, Feb 1, 2000. Write your name, id and the **Tirgul group number** on top of the first page. Keep a copy of the homework. Please answer all questions that require you to write a procedure by submitting the procedure itself, and the result of testing your code on the computer (a printout). Try to pick at least one meaningful example of an execution to show that the procedure works. There is a lot of work so we suggest that you start working right away.

The Meta-Circular Evaluator

1. Write down the necessary changes to the various parts of the evaluator code so that our representation of a frame is of the association list form `('frame (a 1) (b 2) (c 3) (d 4))` instead of the current form that is a cons of two lists `((a b c d).(1 2 3 4))`.

2. It is somewhat curious that we are implementing our SCHEME interpreter in SCHEME. This need not be the case. *Metalinguistic abstraction* is the process of defining a new language given an existing language. We could as well implement in SCHEME an interpreter for some other language.

Change our evaluator so that `define` expressions in our new language can also have the syntactic form

```
(variable <- expression)
```

3. Modify the evaluator to handle the special forms `and` and `or`. This should be done by defining appropriate syntax procedures, and evaluation procedures `eval-and` and `eval-or`.

The value returned for `and` (respectively `or`) is as follows: The subexpressions are evaluated from left to right. The value of the first subexpression that evaluates to false (a true value), is returned and the rest of the subexpressions are not evaluated. If all subexpressions evaluate to a true value (false), the value of the last subexpression is returned. If there are no subexpressions, true (false) is returned.

4. What would happen if we added the new special form for `or` right before the `(else (error ...))` clause in `eval`?

5. Note that there are no reserved words in the language we have implemented. Nothing prevents us from defining or binding (as a parameter) the name of a special form; but we cannot get access to our definition by using the name as an operator. Consider the following example.

```
(define (foo cond else)
  (cond ((= cond 4) 0)
        (else (else cond))))
(define cond 2)
(define (else x) (* x x))
```

After evaluating the definition above, what will our interpreter give for each of the following expressions?

```
==> (foo 4 square)
==> (foo 2 square)
==> (cond ((= cond 4) 0)
          (else (else 5)))
```

6. Solve exercise 4.6 on page 375 of SICP2.

7. This question appeared in a test: We would like to add to the language the following new special form:

```
(repeat <times-exp> <body-exp>)
```

with the following semantics (meaning): The expression `<times-exp>` is evaluated and then the expression `<body-exp>` is evaluated N times, where N is the value returned by `<times-exp>`. (We assume `<times-exp>` returns a non-negative integer value). The value returned by the whole expression is the symbol `ok`. For example,

```
;; M-Eval input:
(define x 5)
;; M-Eval value:
ok
;; M-Eval input:
(repeat (+ x 3) (set! x (+ x 1)))
;; M-Eval value:
ok
;; M-Eval input:
x
;; M-Eval value:
13
```

Modify the evaluator to implement the `repeat` special form.