Assignment 3

April 23, 1996

1 Due Date

April 30, 1996. Please turn in your solution at the beginning of the lecture.

2 The Source Language

The concrete syntax of the language is irrelevant.[1] The abstract syntax trees T are defined inductively as follows. The syntactic category Zs refers to a list of Z.

$$
\begin{array}{rcll}
T & ::= & \mathsf{lit}\ (n) & \text{(Integer Literals)} \\
 & | & \mathsf{varref}\ (x) & \text{(Variables)} \\
 & | & \mathsf{if}\ (T\ T\ T) & \text{(Conditionals)} \\
 & | & \mathsf{let}\ (Ds\ T) & \text{(Blocks)} \\
 & | & \mathsf{varassign}\ (x\ T) & \text{(Assignments)} \\
 & | & \mathsf{begin}\ (T\ T) & \text{(Sequencing)} \\
 & | & \mathsf{proc}\ (xs\ T) & \text{(User-defined procedures)} \\
 & | & \mathsf{app}\ (T\ Ts) & \text{(Applications)} \\
 & | & \mathsf{letrec}\ (Ds\ T) & \text{(Recursive Declarations)}
\end{array}
$$

$$
D \ ::= \ \mathsf{decl}\ (x\ T) \qquad \text{(Declaration)}
$$

More specifically the abstract syntax is defined using the following Scheme declarations:

```
(define-record lit (datum))
(define-record varref (var))
(define-record if (test-exp then-exp else-exp))
(define-record let (decls body))
(define-record varassign (var exp))
(define-record begin (exp1 exp2))
(define-record proc (formals body))
(define-record app (rator rands))
(define-record letrec (decls body))

(define-record decl (var exp))
```

Some of the variables are bound in the original environment to primitive procedures:

$$
P \ ::= \ + \mid - \mid * \mid \mathsf{add1} \mid \mathsf{sub1} \mid \mathsf{error} \qquad \text{(Primitives)}
$$

[1] The book describes two possible concrete syntaxes for the language. Choose the one that you like for testing your solution.

The identifiers $+$, $-$, $*$, add1, and sub1 are bound to the obvious corresponding procedures. The identifier error is bound to a procedure of one argument that aborts the execution of the program, and returns its argument as the final answer. For example,

```
--> add1(add1(error(sub1(3))));
2
```

3 Basic Interpreter

Write a function eval-exp that takes an abstract syntax tree and evaluates it. Hint: Represent proc using Scheme's **lambda**, implement varassign and letrec using assignments, and implement error using *call/cc*.

4 Challenges

Implement as many of the following interpreters as you can. Typically each new interpreter will require some small modifications to your original solution.

- Assuming environments are represented as functions, implement letrec without using assignments.
- Without any assumptions about the representation of environments, implement letrec without using assignments.
- implement varassign without using any assignments.
- implement error without using *call/cc*.