**Interpreter after adding if and let**

**(define-datatype proc-val proc-val?**

**[prim-proc**

**(name symbol?)])**

**(define top-level-eval**

**(lambda (form)**

**(eval-exp form** (empty-env)**)))**

**(define eval-exp**

**(lambda (exp** env**)**

**(cases expression exp**

**[lit-exp (datum) datum]**

**[var-exp (id)**

**(apply-env** env **id**

**(lambda (x) x)** *; procedure to call if var is in env*

(lambda () *; procedure to call if var is not in env*

(apply-env global-env id

(lambda (x) x)

(lambda ()   
 (error 'apply-env "variable ~s is not bound"   
 id))))**)]**

**[app-exp (rator rands)**

**(let ([proc-value (eval-exp rator** env**)]**

**[args (eval-rands rands** env**)])**

**(apply-proc proc-value args))]**

**[if-exp (test true false) (if (eval-exp test** env**)**

**(eval-exp true** env**)**

**(eval-exp false** env**))]**

[let-exp (syms vals bodies)

(let ([extended-env

(extend-env syms

(map (lambda (x)(eval-exp x env))

vals)

env)])

(let loop ([bodies bodies])

(if (null? (cdr bodies))

(eval-exp (car bodies) extended-env)

(begin (eval-exp (car bodies) extended-env)

(loop (cdr bodies))))))]

**[else (error 'eval-exp "Bad abstract syntax: ~a" exp)])))**

**(define apply-proc**

**(lambda (proc-value args)**

**(cases proc-val proc-value**

**[prim-proc (op) (apply-prim-proc op args)]**

**[else (error 'apply-proc**

**"Attempt to apply bad procedure: ~s"**

**proc-value)])))**