MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Spring 2004

Recitation 26
Explicit Control Eval

```
read-eval-print-loop
  (perform (op initialize-stack))
  (perform
   (op prompt-for-input)
   (const ";;; EC-Eval input:"))
  (assign exp (op read))
  (assign env (op get-global-environment))
  (assign continue (label print-result))
  (goto (label eval-dispatch))
print-result
  (perform (op print-stack-statistics))
  (perform
   (op announce-output)
   (const ";;; EC-Eval value:"))
  (perform (op user-print) (reg val))
  (goto (label read-eval-print-loop))

unknown-expression-type
  (assign val
        (const unknown-expression-type-error))
  (goto (label signal-error))
unknown-procedure-type
  (restore continue)
  (assign val
        (const unknown-procedure-type-error))
  (goto (label signal-error))

signal-error
  (perform (op user-print) (reg val))
  (goto (label read-eval-print-loop))

eval-dispatch
  (test (op self-evaluating?) (reg exp))
  (branch (label ev-self-eval))
  (test (op variable?) (reg exp))
  (branch (label ev-variable))
  (test (op quoted?) (reg exp))
  (branch (label ev-quoted))
  (test (op assignment?) (reg exp))
  (branch (label ev-assignment))
  (test (op definition?) (reg exp))
  (branch (label ev-definition))
  (test (op if?) (reg exp))
  (branch (label ev-if))
  (test (op lambda?) (reg exp))
  (branch (label ev-lambda))
  (test (op begin?) (reg exp))
  (branch (label ev-begin))
  (test (op application?) (reg exp))
  (branch (label ev-application))
  (goto (label unknown-expression-type))

ev-self-eval
  (assign val (reg exp))
  (goto (reg continue))
ev-variable
  (assign val (op lookup-variable-value)
         (reg exp) (reg env))
  (goto (reg continue))
ev-quoted
  (assign val (op text-of-quotation) (reg exp))
  (goto (reg continue))
ev-lambda
  (assign unev (op lambda-parameters) (reg exp))
  (assign exp (op lambda-body) (reg exp))
  (assign val (op make-procedure)
              (reg unev) (reg exp) (reg env))
  (goto (reg continue))

ev-application
  (save continue)
  (save env)
  (assign unev (op operands) (reg exp))
  (save unev)
  (assign exp (op operator) (reg exp))
  (assign continue (label ev-appl-did-operator))
  (goto (label eval-dispatch))
ev-appl-did-operator
  (restore unev)
  (restore env)
  (assign argl (op empty-arglist))
  (assign proc (reg val))
  (test (op no-operands?) (reg unev))
  (branch (label apply-dispatch))
  (save proc)
ev-appl-operand-loop
  (save argl)
  (assign exp (op first-operand) (reg unev))
  (test (op last-operand?) (reg unev))
  (branch (label ev-appl-last-arg))
  (save env)
  (save unev)
  (assign continue (label ev-appl-accumulate-arg))
  (goto (label eval-dispatch))
ev-appl-accumulate-arg
  (restore unev)
  (restore env)
  (restore argl)
  (assign argl (op adjoin-arg) (reg val) (reg argl))
  (assign unev (op rest-operands) (reg unev))
  (goto (label ev-appl-operand-loop))

ev-appl-last-arg
  (assign continue (label ev-appl-accum-last-arg))
  (goto (label eval-dispatch))
ev-appl-accum-last-arg
  (restore argl)
  (assign argl (op adjoin-arg) (reg val) (reg argl))
  (restore proc)
  (goto (label apply-dispatch))
apply-dispatch
  (test (op primitive-procedure?) (reg proc))
  (branch (label primitive-apply))
  (test (op compound-procedure?) (reg proc))
  (branch (label compound-apply))
  (goto (label unknown-procedure-type))

primitive-apply
  (assign val (op apply-primitive-procedure)
              (reg proc)
              (reg argl))
  (restore continue)
  (goto (reg continue))

compound-apply
  (assign unev (op procedure-parameters) (reg proc))
  (assign env (op procedure-environment) (reg proc))
  (assign env (op extend-environment)
             (reg unev) (reg argl) (reg env)))
  (assign unev (op procedure-body) (reg proc)))
  (goto (label ev-sequence))

ev-begin
  (assign unev (op begin-actions) (reg exp))
  (save continue)
  (goto (label ev-sequence))

ev-sequence
  (assign exp (op first-exp) (reg unev))
  (test (op last-exp?) (reg unev))
  (branch (label ev-sequence-last-exp))
  (save unev)
  (save env)
  (assign continue (label ev-sequence-continue))
  (goto (label eval-dispatch))
ev-sequence-continue
  (restore env)
  (restore unev)
  (assign unev (op rest-exps) (reg unev))
  (goto (label ev-sequence))
ev-sequence-last-exp
  (restore continue)
  (goto (label eval-dispatch))

ev-if
  (save exp)
  (save env)
  (save continue)
  (assign continue (label ev-if-decide))
  (assign exp (op if-predicate) (reg exp))
  (goto (label eval-dispatch))
ev-if-decide
  (restore continue)
  (restore env)
  (restore exp)
  (test (op true?) (reg val))
  (branch (label ev-if-consequent))
ev-if-alternative
  (assign exp (op if-alternative) (reg exp))
  (goto (label eval-dispatch))
ev-if-consequent
  (assign exp (op if-consequent) (reg exp))
  (goto (label eval-dispatch))

ev-assignment
  (assign unev (op assignment-variable) (reg exp))
  (save unev)
  (assign exp (op assignment-value) (reg exp))
  (save env)
  (save continue)
  (assign continue (label ev-assignment-1))
  (goto (label eval-dispatch))

ev-assignment-1
  (restore continue)
  (restore env)
  (restore unev)
  (perform
   (op set-variable-value!) (reg unev) (reg val)
           (reg env))
  (assign val (const ok))
  (goto (reg continue))

ev-definition
  (assign unev (op definition-variable) (reg exp))
  (save unev)
  (assign exp (op definition-value) (reg exp))
  (save env)
  (save continue)
  (assign continue (label ev-definition-1))
  (goto (label eval-dispatch))
ev-definition-1
  (restore continue)
  (restore env)
  (restore unev)
  (perform
   (op define-variable!) (reg unev) (reg val)
          (reg env))
  (assign val (const ok))
  (goto (reg continue))
```

Adding let

```
ev-let
      (save continue)
      -----------
      (assign argl (op empty-arglist))
      (assign proc (op binding-names) (reg exp))
      ----------
      (assign unev (op binding-values) (reg exp))
ev-let-operand-loop
      (save argl)
      (assign exp (op first-binding) (reg unev))
      (test (op last-operand?) (reg unev))
      (branch (label ev-let-last-binding))
      (save env)
      (save unev)
      (assign continue (label ev-let-accumulate-binding
      (goto (label eval-dispatch))

ev-let-accumulate-binding
      (restore unev)
      (restore env)
      (restore argl)
      (assign argl (op adjoin-arg) (reg val) (reg argl)
      (assign unev (op rest-operands) (reg unev))
      (goto (label ev-let-operand-loop))

ev-let-last-binding
      (assign continue (label ev-let-accum-last-binding
      (goto (label eval-dispatch))
   ev-let-accum-last-binding
      (restore argl)
      (assign argl (op adjoin-arg) (reg val) (reg argl)
      (restore proc)
      (goto (label ev-final-let))

ev-final-let
      (restore exp)
      (assign env (op extend-environment)
                   ___________________________
      (assign unev (op let-body) ___________)
      (goto (label ev-sequence))
```

1. By reference to the rest of EC-eval, and knowledge of the desugaring of let, where was most of this code appropriated from? (It was copied with label modification from eceval)
2. Fill in the blanks.
3. There's a bug in this code, can you spot it? (Hint: it has to do with contracts).