The *Function* Eval

Evaluation is an implicit action of the ALU. By claiming evaluation is automatic, we are committed to wiring the ALU in a specific way. However handling memory can be made flexible by *defining* **Eval** *in the programming language itself*. This process is called *meta-circular evaluation,* cause it uses a language itself to define how that language should behave. All we have to do is to define the evaluation function by telling the system what to do when an expression is typed in. The function **Eval** takes two arguments, the expression to be evaluated and the *binding environment*, that is, an address of the memory array which contains all of the primitive functions and atoms (and any other symbols which we may have added) in the language. The binding environment contains the definitions of all user defined functions, and the values of each of the variables (function arguments).

Since the binding environment does not change in this example, (i.e. we have not designed the language to establish separate environments for each function call), we will treat the token **Eval** to mean "Eval-in-environment". (Some of the syntax has been changed to make **Eval** more readable.)

The definition of **Eval** which follows recognizes only seven reserved words as primitive functions. In addition, **Eval** uses three built-in tests to determine the types of objects.

First	Rest	Cons	
IfThenElse	Equal	Quote	Let
Is-empty	Isa-atom	Isa-expression	

[Notes and supporting functions are on this page to save space. **Eval** itself is on the next page.]

Notes, * process Atom in First: This defines a syntax for parsing. Every expression begins with an atom or is an atom. If an expression begins with an atom, the processor assumes that that atom is an operator, and thus a processing instruction. The operator **Quote** is the no-op.

Notes, ** Cons Eval of Rest: This is again a syntax constraint. Once we have removed the beginning operator of an expression, what follows is either an atom, or another expression which itself begins with an atom operator.

```
EvalLogic exp  =def=
      If Equal (Eval (First exp)) (Quote True)          ;if First is TRUE
       Then                                             ;Eval second argument
         Eval (Rest exp)
       Else                                             ;Eval third argument
         Eval (Rest (Rest exp)))

EvalExp exp   =def=
      If Is-empty exp                                   ;if at the end
       Then                                             ;return ground
         nil
       Else                                             ;Eval the parts
        Cons (Eval (First exp)) (Eval (Rest exp))       ; and put them together
```

```
Eval exp  =def=

If Isa-atom exp
 Then                                                    ;process atom
   If Is-empty (First exp)
    Then                                                 ;return the SYMBOL
      Rest exp
    Else                                                 ; or its VALUE
      First exp
 Else                                                    ;process expression
   If Isa-atom (First exp)
    Then                                                 ;process Atom in First*
      Let token (First exp)                              ;naming the atom
        If Equal token (Quote Quote)
         Then                                            ;return what follows
           Rest exp
         Else                                              ;other operators
           If Equal token (Quote IfThenElse)
            Then                                         ;process logic operator
              EvalLogic (Rest exp)
            Else                                           ;other operators
              If Equal token (Quote First)
               Then                                      ;First of Eval of Rest
                 First (Eval (Rest exp))
               Else                                        ;other operators
                 If Equal token (Quote Rest)
                  Then                                   ;Rest of Eval of Rest
                    Rest (Eval (Rest exp))
                  Else                                     ;other operators
                    If Equal token (Quote Isa-atom)
                     Then                                ;Isa-atom Eval of Rest
                       Isa-atom (Eval (Rest exp))
                     Else                                  ;other operators
                       If Isa-expression token
                        Then                             ;process expression
                          EvalExp (Rest exp)
                        Else                               ;other operators
                          If Equal token (Quote Cons)
                           Then                          ;Cons Eval of Rest**
                             Cons (Eval (Rest exp))
                                  (Eval (Rest (Rest exp)))
                           Else                            ;other operators
                             If Equal token (Quote Equal)
                              Then                       ;Equal Eval of args
                                Equal (Eval (Rest exp))
                                      (Eval (Rest (Rest exp)))
                              Else                       ;replace the token
                                Eval                     ; with its value
                                   Cons (First token) (Rest exp)
   Else ERROR))
```