```
' SBLisp version by John Spikowski and Mike Lobanovsky - Aug. 8th,
2014
' Bitbucket Repository: https://bitbucket.org/ScriptBasic/sblisp
'
' This work is licensed under the Creative Commons
' Attribution-NonCommercial-ShareAlike License. To view a copy of this
' license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/ or
' send a letter to Creative Commons, 171 Second St, Suite 300
' San Francisco, CA 94105, USA.
'
' The original author of this code is Arthur Nunes-Harwitt

' If not using basic.conf, add quotes around error.bas including
' any path if not in the same directory as lisp.sb

IMPORT error.bas

ON ERROR GOTO HandleInternalError

cmdln = COMMAND()

SPLITA TRIM(cmdln) BY " " TO cmdlnargs
dbgflg = FALSE
FOR x = 0 TO UBOUND(cmdlnargs)
  IF TRIM(UCASE(cmdlnargs[x])) = "-D" THEN
    dbgflg = TRUE
  ELSE IF TRIM(cmdlnargs[x]) <> undef THEN
    cmdflg = TRUE
    cmdlnfn = TRIM(cmdlnargs[x])
  ELSE
    cmdflg = FALSE
  END IF
NEXT


'-------
' SUBs
'-------

SUB LispOpenInputFile
  OPEN lispfilename FOR INPUT AS lispfilenum
  bsd -= 1
END SUB
```

```
SUB LispCloseFile
  CLOSE lispfilenum
  lispfilename = ""
  lispfilenum = 0
  bsd -= 1
END SUB

SUB GetSymbol
  total = 0
  opos = ipos
  smb = 1
  slength = FALSE
GetHashNumLoop:
  IF ipos = LEN(ibuf) + 1 THEN GOTO CheckExistance
  curchar = UCASE(MID(ibuf, ipos, 1))
  IF slength = TRUE THEN
    IF curchar = "\"" THEN
      ipos += 1
      GOTO CheckExistance
    ELSE
      GOTO DoStrLiteral
    END IF
  ELSE
    IF curchar = "\"" THEN
      ipos += 1
      opos = ipos
      slength = TRUE
    END IF
  END IF
  IF INSTR(" ()'", curchar) <> undef THEN GOTO CheckExistance
DoStrLiteral:
  temp = ASC(curchar) * smb + total
  total = temp - (INT(temp / maxsymboltablesize) * maxsymboltablesize
  )
  temp = smb * 256
  smb = temp - (INT(temp / maxsymboltablesize) * maxsymboltablesize)
  ipos += 1
  GOTO GetHashNumLoop
CheckExistance:
  IF symbols[total] = CHR(0) THEN GOTO PutInTable
  temp = symbols[total]
  IF temp = UCASE(MID(ibuf, opos, ipos - opos + slength)) THEN
```

```
    ctype = symbol
    cvalue = total
    bsd -= 1
    EXIT SUB
  END IF
  temp = total * total
  total = temp - (INT(temp / maxsymboltablesize) * maxsymboltablesize
  )
  GOTO CheckExistance
PutInTable:
  IF slotsfilled = maxsymboltablesize THEN
    PRINT "ERROR: Symbol table full.\n"
    END
  END IF
  symbols[total] = UCASE(MID(ibuf, opos, ipos - opos + slength))
  ctype = symbol
  cvalue = total
  slotsfilled += 1
  bsd -= 1
END SUB

SUB RregtoPreg
  ptype = rtype
  pvalue = rvalue
  bsd -= 1
END SUB

SUB RregtoQreg
  qtype = rtype
  qvalue = rvalue
  bsd -= 1
END SUB

SUB CopyPair
  IF heaptypeactive[gvalue] = brokenheart THEN
    heapvaluepassive[scan] = heapvalueactive[gvalue]
    bsd -= 1
    EXIT SUB
  END IF
  heaptypepassive[free] = heaptypeactive[gvalue - 1]
  heapvaluepassive[free] = heapvalueactive[gvalue - 1]
  free += 1
  heaptypepassive[free] = heaptypeactive[gvalue]
```

```
  heapvaluepassive[free] = heapvalueactive[gvalue]
  heaptypeactive[gvalue] = brokenheart
  heapvalueactive[gvalue] = free
  heapvaluepassive[scan] = free
  free += 1
  bsd -= 1
END SUB

SUB CopyProcTriple
  IF heaptypeactive[gvalue] = brokenheart THEN
    heapvaluepassive[scan] = heapvalueactive[gvalue]
    bsd -= 1
    EXIT SUB
  END IF
  heaptypepassive[free] = heaptypeactive[gvalue - 2]
  heapvaluepassive[free] = heapvalueactive[gvalue - 2]
  free += 1
  heaptypepassive[free] = heaptypeactive[gvalue - 1]
  heapvaluepassive[free] = heapvalueactive[gvalue - 1]
  free += 1
  heaptypepassive[free] = heaptypeactive[gvalue]
  heapvaluepassive[free] = heapvalueactive[gvalue]
  heaptypeactive[gvalue] = brokenheart
  heapvalueactive[gvalue] = free
  heapvaluepassive[scan] = free
  free += 1
  bsd -= 1
END SUB

SUB StartCopyPair
  IF heaptypeactive[gvalue] = brokenheart THEN
    bsd -= 1
    EXIT SUB
  END IF
  heaptypepassive[free] = heaptypeactive[gvalue - 1]
  heapvaluepassive[free] = heapvalueactive[gvalue - 1]
  free += 1
  heaptypepassive[free] = heaptypeactive[gvalue]
  heapvaluepassive[free] = heapvalueactive[gvalue]
  heaptypeactive[gvalue] = brokenheart
  heapvalueactive[gvalue] = free
  free += 1
  bsd -= 1
```

```
END SUB

SUB StartCopyTriple
  IF heaptypeactive[gvalue] = brokenheart THEN
    bsd -= 1
    EXIT SUB
  END IF
  heaptypepassive[free] = heaptypeactive[gvalue - 2]
  heapvaluepassive[free] = heapvalueactive[gvalue - 2]
  free += 1
  heaptypepassive[free] = heaptypeactive[gvalue - 1]
  heapvaluepassive[free] = heapvalueactive[gvalue - 1]
  free += 1
  heaptypepassive[free] = heaptypeactive[gvalue]
  heapvaluepassive[free] = heapvalueactive[gvalue]
  heaptypeactive[gvalue] = brokenheart
  heapvalueactive[gvalue] = free
  free += 1
  bsd -= 1
END SUB

SUB MakeNumber
  IF decimalp THEN decimal -= 1
  ctype = number
  cvalue = numsign * total / (10 ^ decimal)
  bsd -= 1
END SUB

SUB NilPrint
  PRINT "F"
  bsd -= 1
END SUB

SUB TPrint
  IF pvalue THEN
    PRINT "T"
    bsd -= 1
    EXIT SUB
  END IF
  PRINT "ERROR: Output unprintable.\n"
  bsd -= 1
END SUB
```

```
'------------
' FUNCTIONs
'------------

FUNCTION PushStack
  stacktype[stkcursor] = ptype
  stackvalue[stkcursor] = pvalue
  stkcursor += 1
  IF stkcursor > maxstacksize THEN
    PRINT "ERROR: Stack overflow.\n"
    PushStack = FALSE
    EXIT FUNCTION
  ELSE
    PushStack = TRUE
  END IF
  bsd -= 1
END FUNCTION

FUNCTION TopOfStack
  IF stkcursor = 0 THEN
    PRINT "ERROR: Stack underflow.\n"
    TopOfStack = FALSE
    EXIT FUNCTION
  ELSE
    TopOfStack = TRUE
  END IF
  rtype = stacktype[stkcursor - 1]
  rvalue = stackvalue[stkcursor - 1]
  bsd -= 1
END FUNCTION

FUNCTION Car
  IF ptype <> pair THEN
    PRINT "ERROR: Bad type in car.\n"
    Car = FALSE
    EXIT FUNCTION
  ELSE
    Car = TRUE
  END IF
  UnTypedCar:
  rtype = heaptypeactive[pvalue]
  rvalue = heapvalueactive[pvalue]
```

```
  bsd -= 1
END FUNCTION

FUNCTION SetCar
  IF ptype <> pair THEN
    PRINT "ERROR: Bad type.\n"
    SetCar = FALSE
    EXIT FUNCTION
  ELSE
    SetCar = TRUE
  END IF
  heaptypeactive[pvalue] = qtype
  heapvalueactive[pvalue] = qvalue
  rtype = qtype
  rvalue = qvalue
  bsd -= 1
END FUNCTION

FUNCTION Cdr
  IF ptype <> pair THEN
    PRINT "ERROR: Bad type in cdr.\n"
    Cdr = FALSE
    EXIT FUNCTION
  ELSE
    Cdr = TRUE
  END IF
  rtype = heaptypeactive[pvalue - 1]
  rvalue = heapvalueactive[pvalue - 1]
  bsd -= 1
END FUNCTION

FUNCTION TestThenScan
StartCopyUsefulMemory:
  IF scan >= free THEN
    bsd -= 1
    TestThenScan = TRUE
    EXIT FUNCTION
  END IF
  gtype = heaptypepassive[scan]
  gvalue = heapvaluepassive[scan]
  IF gtype = pair THEN
    bsd += 1
    CopyPair()
```

```
  END IF
  IF gtype = procedure THEN
    bsd += 1
    CopyProcTriple()
  END IF
  IF scan > maxheapsize OR free > maxheapsize THEN
    PRINT "ERROR: Out of memory.\n"
    SWAP heaptype0, heaptype1
    SWAP heapvalue0, heapvalue1
    CopyUsefulMemory = FALSE
    EXIT FUNCTION
  ELSE
    CopyUsefulMemory = TRUE
  END IF
  IF gtype = brokenheart THEN
    PRINT "A broken heart.\n"
    END
  END IF
  scan += 1
  GOTO StartCopyUsefulMemory
END FUNCTION

FUNCTION PopStack
  IF stkcursor = 0 THEN
    PRINT "ERROR: Stack Underflow.\n"
    PopStack = FALSE
    EXIT FUNCTION
  ELSE
    PopStack = TRUE
  END IF
  stkcursor -= 1
  bsd -= 1
END FUNCTION

FUNCTION SetCdr
  IF ptype <> pair THEN
    PRINT "ERROR: Bad type.\n"
    SetCdr = FALSE
    EXIT FUNCTION
  ELSE
    SetCdr = TRUE
  END IF
  heaptypeactive[pvalue - 1] = qtype
```

```
  heapvalueactive[pvalue - 1] = qvalue
  rtype = qtype
  rvalue = qvalue
  bsd -= 1
END FUNCTION


'-------
' MAIN
'-------

InitMemory:
  maxheapsize=20000

  SPLITA STRING(maxheapsize + 1, "") BY "" TO heaptypeactive
  SPLITA STRING(maxheapsize + 1, "") BY "" TO heaptypepassive
  SPLITA STRING(maxheapsize + 1, "") BY "" TO heapvalueactive
  SPLITA STRING(maxheapsize + 1, "") BY "" TO heapvaluepassive

  ref heaptype0 = heaptypeactive
  ref heaptype1 = heaptypepassive
  ref heapvalue0 = heapvalueactive
  ref heapvalue1 = heapvaluepassive

  maxsymboltablesize = 2000
  SPLITA STRING(maxsymboltablesize + 1, 0) BY "" TO symbols
  slotsfilled = 0

  maxstacksize = 10000
  SPLITA STRING(maxstacksize + 1, "") BY "" TO stacktype
  SPLITA STRING(maxstacksize + 1, "") BY "" TO stackvalue

  hpcursor = 0
  stkcursor = 0
  temp = 0
  ibuf = ""
  ipos = 1
  curchar = ""
  numevals = 0
  bsd = 0
  ctype = 0
  cvalue = 0
  rtype = 0
```

```
  rvalue = 0
  ptype = 0
  pvalue = 0
  qtype = 0
  qvalue = 0
  btype = 0
  bvalue = 0
  dtype = 0
  dvalue = 0
  etype = 0
  evalue = 0
  truthtest = FALSE
  done = FALSE
  lispfilename = ""
  lispfilenum = 0
  brokenheart = -2
  actsym = -1
  naught = 0
  booleant = 1
  primitive = 2
  number = 3
  symbol = 4
  pair = 5
  procedure = 6
  lp = 0
  rp = 1
  quotes = 2
  dot = 3
  oparen = 0

InitLispEnv:
  ibuf = "DEFINE"
  ipos = 1
  bsd += 1
  GetSymbol()
  scmdef = cvalue

  ibuf = "LAMBDA"
  ipos = 1
  bsd += 1
  GetSymbol()
  scmlambda = cvalue
```

```
  ibuf = "QUOTE"
  ipos = 1
  bsd += 1
  GetSymbol()
  scmquote = cvalue

  ibuf = "COND"
  ipos = 1
  bsd += 1
  GetSymbol()
  scmcond = cvalue

  ibuf = "IF"
  ipos = 1
  bsd += 1
  GetSymbol()
  scmif = cvalue

  ibuf = "SET!"
  ipos = 1
  bsd += 1
  GetSymbol()
  scmset = cvalue

  ibuf = "ELSE"
  ipos = 1
  bsd += 1
  GetSymbol()
  scmelse = cvalue

  ibuf = "AND"
  ipos = 1
  bsd += 1
  GetSymbol()
  scmand = cvalue

  ibuf = "OR"
  ipos = 1
  bsd += 1
  GetSymbol()
  scmor = cvalue

  ibuf = "SEQUENCE"
```

```
  ipos = 1
  bsd += 1
  GetSymbol()
  scmseq = cvalue

  ibuf = "LET"
  ipos = 1
  bsd += 1
  GetSymbol()
  scmlet = cvalue

  ibuf = "LET*"
  ipos = 1
  bsd += 1
  GetSymbol()
  scmletstar = cvalue

  primitives[0,0] = ""
  primitives[0,1] = 0
  primitives[1,0] = "CONS"
  primitives[1,1] = 1
  primitives[2,0] = "CAR"
  primitives[2,1] = 2
  primitives[3,0] = "CDR"
  primitives[3,1] = 3
  primitives[4,0] = "EQ?"
  primitives[4,1] = 4
  primitives[5,0] = "NULL?"
  primitives[5,1] = 5
  primitives[6,0] = "NOT"
  primitives[6,1] = 5
  primitives[7,0] = "="
  primitives[7,1] = 6
  primitives[8,0] = "+"
  primitives[8,1] = 7
  primitives[9,0] = "-"
  primitives[9,1] = 8
  primitives[10,0] = "*"
  primitives[10,1] = 9
  primitives[11,0] = "/"
  primitives[11,1] = 10
  primitives[12,0] = "QUIT"
  primitives[12,1] = 11
```

```
  primitives[13,0] = "NUMBER?"
  primitives[13,1] = 12
  primitives[14,0] = "SYMBOL?"
  primitives[14,1] = 13
  primitives[15,0] = "PAIR?"
  primitives[15,1] = 14
  primitives[16,0] = "PROCEDURE?"
  primitives[16,1] = 15
  primitives[17,0] = "BOUND?"
  primitives[17,1] = 16
  primitives[18,0] = "ASSQ"
  primitives[18,1] = 17
  primitives[19,0] = "REVERSE"
  primitives[19,1] = 18
  primitives[20,0] = "LIST"
  primitives[20,1] = 19
  primitives[21,0] = "READ"
  primitives[21,1] = 20
  primitives[22,0] = "PRINT"
  primitives[22,1] = 21
  primitives[23,0] = "NEWLINE"
  primitives[23,1] = 22
  primitives[24,0] = "SET-CAR!"
  primitives[24,1] = 23
  primitives[25,0] = "SET-CDR!"
  primitives[25,1] = 24
  primitives[26,0] = ">="
  primitives[26,1] = 25
  primitives[27,0] = "<="
  primitives[27,1] = 26
  primitives[28,0] = ">"
  primitives[28,1] = 27
  primitives[29,0] = "<"
  primitives[29,1] = 28
  primitives[30,0] = "EXP"
  primitives[30,1] = 29
  primitives[31,0] = "LOG"
  primitives[31,1] = 30
  primitives[32,0] = "FLOOR"
  primitives[32,1] = 31
  primitives[33,0] = "MEMQ"
  primitives[33,1] = 32
  primitives[34,0] = "LENGTH"
```

```
  primitives[34,1] = 33
  primitives[35,0] = "EQUAL?"
  primitives[35,1] = 34
  primitives[36,0] = "MEMBER"
  primitives[36,1] = 35
  primitives[37,0] = "ASSOC"
  primitives[37,1] = 36
  primitives[38,0] = "SIN"
  primitives[38,1] = 37
  primitives[39,0] = "COS"
  primitives[39,1] = 38
  primitives[40,0] = "ATAN"
  primitives[40,1] = 39
  primitives[41,0] = "GC"
  primitives[41,1] = 40
  primitives[42,0] = "EVAL"
  primitives[42,1] = 41
  primitives[43,0] = "APPLY"
  primitives[43,1] = 42
  primitives[44,0] = "FOR-EACH"
  primitives[44,1] = 43
  primitives[45,0] = "MAP"
  primitives[45,1] = 44
  primitives[46,0] = "LOAD"
  primitives[46,1] = 45

  maxprim = 46
  ibuf = "T"
  ipos = 1
  bsd += 1
  GetSymbol()
  ptype = ctype
  pvalue = cvalue
  qtype = booleant
  qvalue = TRUE
  bsd += 1
  GOSUB Cons
  bsd += 1
  RregtoPreg()
  qtype = naught
  qvalue = FALSE
  bsd += 1
  GOSUB Cons
```

```
  initenv = rvalue
  ibuf = "NIL"
  ipos = 1
  bsd += 1
  GetSymbol()
  ptype = ctype
  pvalue = cvalue
  qtype = naught
  qvalue = FALSE
  bsd += 1
  GOSUB Cons
  bsd += 1
  RregtoPreg()
  qtype = pair
  qvalue = initenv
  bsd += 1
  GOSUB Cons
  initenv = rvalue
  FOR prm = 1 TO maxprim
    ibuf = primitives[prm, 0]
    primnum = primitives[prm, 1]
    ipos = 1
    bsd += 1
    GetSymbol()
    ptype = ctype
    pvalue = cvalue
    qtype = primitive
    qvalue = primnum
    bsd += 1
    GOSUB Cons
    bsd += 1
    RregtoPreg()
    qtype = pair
    qvalue = initenv
    bsd += 1
    GOSUB Cons
    initenv = rvalue
  NEXT prm
  bsd += 1
  RregtoPreg()
  qtype = naught
  qvalue = FALSE
  bsd += 1
```

```
  GOSUB Cons
  initenv = rvalue
  globalenv = initenv

PRINT "SBLisp - Scheme BASIC Lisp\n\n"

LispReadEvalPrintLoop:
  stkcursor = 0
  curenv = globalenv
  bsd += 1
  GOSUB LispRead
  bsd += 1
  RregtoPreg()
  bsd += 1
  GOSUB LispEval
  bsd += 1
  RregtoPreg()
  bsd += 1
  GOSUB LispPrint
  PRINTNL
  GOTO LispReadEvalPrintLoop

DoQit:
  END


'-----------------
' ERROR Handling
'-----------------

HandleInternalError:
  IF ERROR() = &H14 THEN
    ON ERROR GOTO HandleInternalError
    RESUME NEXT
  END IF
  RESUME

HandleError:
  IF lispfilenum <> 0 THEN
    PRINT "ERROR: Problem in file ", lispfilename, "\n"
    bsd += 1
    LispCloseFile()
  END IF
```

```
  ClearStackLoop:
  IF bsd <> 0 THEN
    bsd -= 1
    GOTO ClearStackLoop
  END IF
  GOTO LispReadEvalPrintLoop


'-----------------
' GOSUB Routines
'-----------------

GarbageCollect:
  scan = 0
  free = 0
  IF dbgflg THEN PRINT "GC start "
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = pair
  pvalue = curenv
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = pair
  pvalue = globalenv
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = btype
  pvalue = bvalue
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = dtype
  pvalue = dvalue
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = etype
```

```
  pvalue = evalue
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  FOR stk = 0 TO stkcursor - 1
    gtype = stacktype[stk]
    gvalue = stackvalue[stk]
    IF gtype = pair THEN
      bsd += 1
      StartCopyPair()
      stackvalue[stk] = heapvalueactive[stackvalue[stk]]
      bsd += 1
      IF NOT TestThenScan() THEN GOTO HandleError
    END IF
    IF gtype = procedure THEN
      bsd += 1
      StartCopyTriple()
      stackvalue[stk] = heapvalueactive[stackvalue[stk]]
      bsd += 1
      IF NOT TestThenScan() THEN GOTO HandleError
    END IF
  NEXT stk
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  etype = rtype
  evalue = rvalue
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  dtype = rtype
  dvalue = rvalue
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  ctype = rtype
  cvalue = rvalue
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  btype = rtype
  bvalue = rvalue
```

```
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  qtype = rtype
  qvalue = rvalue
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  globalenv = rvalue
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  curenv = rvalue
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  ptype = rtype
  pvalue = rvalue
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  hpcursor = free
  IF free = maxheapsize THEN
    PRINT "Out of memory.\n"
    END
  END IF
  SWAP heaptype0, heaptype1
  SWAP heapvalue0, heapvalue1
  IF dbgflg THEN PRINT "GC done\n"
  bsd -= 1
  RETURN

Cons:
  IF hpcursor + 2 > maxheapsize THEN
    bsd += 1
    GOSUB GarbageCollect
  END IF
  heaptypeactive[hpcursor] = qtype
  heapvalueactive[hpcursor] = qvalue
  hpcursor += 1
```

```
  heaptypeactive[hpcursor] = ptype
  heapvalueactive[hpcursor] = pvalue
  rtype = pair
  rvalue = hpcursor
  hpcursor += 1
  bsd -= 1
  RETURN

GetLine:
  IF cmdflg THEN
    ibuf = "(load '" & cmdlnfn & ")"
    ipos = 1
    cmdflg = FALSE
    bsd -=1
    RETURN
  END IF
  IF lispfilenum = 0 THEN
    PRINT LTRIM(FORMAT("%~##~",oparen)), "]"
    LINE INPUT ibuf
    ibuf = CHOMP(ibuf)
    compos = INSTR(ibuf, ";")
    IF compos <> undef THEN ibuf = RTRIM(LEFT(ibuf, compos - 1))
    IF ASC(ibuf) = undef THEN GOTO GetLine
    ipos = 1
  ELSE
    IF NOT EOF(lispfilenum) THEN
      LINE INPUT# lispfilenum, ibuf
      ibuf = CHOMP(ibuf)
      compos = INSTR(ibuf, ";")
      IF compos <> undef THEN ibuf = RTRIM(LEFT(ibuf, compos - 1))
      IF ASC(ibuf) = undef THEN GOTO GetLine
      PRINT ibuf, "\n"
      ipos = 1
    ELSE
      LispCloseFile()
      RETURN
    END IF
  END IF
  bsd -= 1
  RETURN

DoLoad:
  ptype = qtype
```

```
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  IF ptype <> symbol THEN
    PRINT "ERROR: Load needs a symbol.\n"
    GOTO HandleError
  END IF
  lispfilename = symbols[pvalue]
  OPTION COMPARE sbCaseInsensitive
  fnpos = INSTR(ibuf, lispfilename)
  IF fnpos THEN lispfilename = MID(ibuf, fnpos, LEN(lispfilename))
  OPTION COMPARE sbCaseSensitive
  lispfilenum = 1
  bsd += 1
  LispOpenInputFile()
  WHILE NOT EOF(lispfilenum)
    stkcursor = 0
    curenv = globalenv
    bsd += 1
    GOSUB LispRead
    bsd += 1
    RregtoPreg()
    bsd += 1
    GOSUB LispEval
    bsd += 1
    RregtoPreg()
    bsd += 1
    GOSUB LispPrint
    PRINTNL
  WEND
  bsd += 1
  LispCloseFile()
  rtype = booleant
  rvalue = TRUE
  bsd -= 1
  RETURN

LispRead:
  oparen = 0
  bsd += 1
  GOSUB GetLine
```

```
ReadSExp:
  IF ipos > LEN(ibuf) THEN
    IF lispfilenum > 0 THEN
      IF EOF(lispfilenum) THEN GOTO HandleError
    END IF
    bsd += 1
    GOSUB GetLine
  END IF
  bsd += 1
  GOSUB GetNextToken
  IF ctype = number OR ctype = symbol THEN
    rtype = ctype
    rvalue = cvalue
    bsd -= 1
    RETURN
  END IF
  IF ctype = actsym AND cvalue = quotes THEN
    bsd += 1
    GOSUB ReadSExp
    bsd += 1
    RregtoPreg()
    qtype = naught
    qvalue = FALSE
    bsd += 1
    GOSUB Cons
    bsd += 1
    RregtoQreg()
    ptype = symbol
    pvalue = scmquote
    GOSUB Cons
    RETURN
  END IF
  IF ctype = actsym AND cvalue = lp THEN GOTO ReadSExpSeq
  IF ctype = actsym AND cvalue = rp THEN
    IF oparen < 0 THEN
      PRINT "ERROR: Too many ).\n"
      GOTO HandleError
    END IF
    rtype = ctype
    rvalue = cvalue
    bsd -= 1
    RETURN
```

```
  END IF
  IF ctype = actsym AND cvalue = dot THEN
    IF oparen < 0 THEN
      PRINT "ERROR: Misplaced . [period].\n"
      GOTO HandleError
    END IF
    rtype = ctype
    rvalue = cvalue
    bsd -= 1
    RETURN
  END IF

LispEval:
  IF ptype = number OR ptype = naught THEN
    rtype = ptype
    rvalue = pvalue
    bsd -= 1
    RETURN
  END IF
  IF ptype = symbol THEN
    GOSUB EvalSymbol
    RETURN
  END IF
  IF ptype = pair THEN
    GOSUB EvalCombination
    RETURN
  END IF
  PRINT "ERROR: Unknown expression.\n"
  GOTO HandleError

LispPrint:
  IF ptype > 6 OR ptype < 0 THEN
    PRINT "ERROR: Bad type.\n"
    GOTO HandleError
    RETURN
  END IF
  IF ptype = 0 THEN
    NilPrint()
    RETURN
  END IF
  IF ptype = 1 THEN
    TPrint()
    RETURN
```

```
  END IF
  IF ptype = 2 THEN
    PRINT ""
    bsd -= 1
    RETURN
  END IF
  IF ptype = 3 THEN
    PRINT STR(pvalue)
    bsd -= 1
    RETURN
  END IF
  IF ptype = 4 THEN
    PRINT symbols[pvalue]
    bsd -= 1
    RETURN
  END IF
  IF ptype = 5 THEN
    PRINT "("
ContinueListPrint:
    ctype = ptype
    cvalue = pvalue
    bsd += 1
    IF NOT Cdr() THEN GOTO HandleError
    bsd += 1
    RregtoPreg()
    bsd += 1
    IF NOT PushStack() THEN GOTO HandleError
    ptype = ctype
    pvalue = cvalue
    bsd += 1
    IF NOT Car() THEN GOTO HandleError
    bsd += 1
    RregtoPreg()
    bsd += 1
    GOSUB LispPrint
    bsd += 1
    IF NOT TopOfStack() THEN GOTO HandleError
    ptype = rtype
    pvalue = rvalue
    bsd += 1
    IF NOT PopStack() THEN GOTO HandleError
    IF ptype = naught THEN
      PRINT ")"
```

```
      bsd -= 1
      RETURN
    END IF
    IF ptype = pair THEN
      PRINT " "
      GOTO ContinueListPrint
    END IF
    PRINT " . "
    bsd += 1
    GOSUB LispPrint
    PRINT ")"
    bsd -= 1
    RETURN
  END IF
  IF ptype = 6 THEN
    PRINT ""
    bsd -= 1
    RETURN
  END IF

GetNextToken:
  decimalp = FALSE
  numsign = 1
  decimal = 0
  IF ipos > LEN(ibuf) THEN
   PRINT "ERROR: Read.\n"
   GOTO HandleError
  END IF
  curchar = MID(ibuf, ipos, 1)
  IF curchar <= " " THEN
    ipos += 1
    GOTO GetNextToken
  END IF
  IF curchar >= "0" AND curchar <= "9" THEN
    opos = ipos
    GOTO GetNumber
  END IF
  IF curchar = "+" OR curchar = "-" THEN
    opos = ipos
    GOTO CheckSign
  END IF
  IF curchar = "." THEN
    opos = ipos
```

```
    GOTO CheckDot
  END IF
  IF curchar = "'" THEN
    ctype = actsym
    cvalue = quotes
    ipos += 1
    bsd -= 1
    RETURN
  END IF
  IF curchar = "(" THEN
    ctype = actsym
    cvalue = lp
    ipos += 1
    oparen += 1
    bsd -= 1
    RETURN
  END IF
  IF curchar = ")" THEN
    ctype = actsym
    cvalue = rp
    ipos += 1
    oparen -= 1
    bsd -= 1
    RETURN
  END IF
  GetSymbol()
  RETURN

ReadSExpSeq:
  bsd += 1
  GOSUB ReadSExp
  IF rtype = actsym AND rvalue = rp THEN
    rtype = naught
    rvalue = FALSE
    bsd -= 1
    RETURN
  END IF
  IF rtype = actsym AND rvalue = dot THEN
    bsd += 1
    GOSUB ReadSExp
    bsd += 1
    RregtoPreg()
    bsd += 1
```

```
    IF NOT PushStack() THEN GOTO HandleError
    bsd += 1
    GOSUB ReadSExp
    IF rtype <> actsym OR rvalue <> rp THEN
      PRINT "ERROR: Badly formed expression.\n"
      GOTO HandleError
    END IF
    bsd += 1
    IF NOT TopOfStack() THEN GOTO HandleError
    IF NOT PopStack() THEN GOTO HandleError
  END IF
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  bsd += 1
  GOSUB ReadSExpSeq
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  GOSUB Cons
  RETURN

GetCarList:
  dtype = rtype
  dvalue = rvalue
  etype = naught
  evalue = FALSE
  WHILE dtype<>naught
    ptype = dtype
    pvalue = dvalue
    bsd += 1
    IF NOT Car() THEN GOTO HandleError
    bsd += 1
    RregtoPreg()
    bsd += 1
    IF NOT Car() THEN GOTO HandleError
    bsd += 1
```

```
    RregtoPreg()
    qtype = etype
    qvalue = evalue
    bsd += 1
    GOSUB Cons
    etype = rtype
    evalue = rvalue
    ptype = dtype
    pvalue = dvalue
    bsd += 1
    IF NOT Cdr() THEN GOTO HandleError
    dtype = rtype
    dvalue = rvalue
  WEND
  rtype = etype
  rvalue = evalue
  bsd -= 1
  RETURN

MapCar:
  btype = ptype
  bvalue = pvalue
  ctype = qtype
  cvalue = qvalue
  done = FALSE
  MapCarReEntry:
  IF done THEN
    rtype=naught
    rvalue = FALSE
    bsd -= 1
    RETURN
  END IF
  ptype = btype
  pvalue = bvalue
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  bsd += 1
  GOSUB GetCarList
  bsd += 1
```

```
  RregtoQreg()
  ptype = btype
  pvalue = bvalue
  bsd += 1
  GOSUB LispApply
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  ctype = rtype
  cvalue = rvalue
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  btype = rtype
  bvalue = rvalue
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  GOSUB GetCdrList
  ctype = rtype
  cvalue = rvalue
  bsd += 1
  GOSUB MapCarReEntry
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  GOSUB Cons
  RETURN

Assq:
  ctype = ptype
  cvalue = pvalue
```

```
  ptype = qtype
  pvalue = qvalue
  AssqLoop:
  IF ptype = naught THEN
    rtype = naught
    rvalue = FALSE
    bsd -= 1
    RETURN
  END IF
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  IF ctype = rtype AND cvalue = rvalue THEN
    rtype = ptype
    rvalue = pvalue
    bsd -= 1
    RETURN
  END IF
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  qtype = ptype
  qvalue = pvalue
  GOTO AssqLoop

EqualP:
  IF ptype <> pair OR qtype <> pair THEN
    IF ptype = qtype AND pvalue = qvalue THEN
      rtype = booleant
      rvalue = TRUE
      bsd -= 1
      RETURN
    ELSE
      rtype=naught
      rvalue = FALSE
```

```
      bsd -= 1
      RETURN
    END IF
  ELSE
    bsd += 1
    IF NOT PushStack() THEN GOTO HandleError
    rtype = ptype
    rvalue = pvalue
    ptype = qtype
    pvalue = qvalue
    bsd += 1
    IF NOT PushStack() THEN GOTO HandleError
    bsd += 1
    RregtoPreg()
    bsd += 1
    IF NOT Car() THEN GOTO HandleError
    ptype = qtype
    pvalue = qvalue
    bsd += 1
    RregtoQreg()
    bsd += 1
    IF NOT Car() THEN GOTO HandleError
    ptype = qtype
    pvalue = qvalue
    bsd += 1
    RregtoQreg()
    bsd += 1
    GOSUB EqualP
    IF rvalue THEN
      bsd += 1
      IF NOT TopOfStack() THEN GOTO HandleError
      bsd += 1
      IF NOT PopStack() THEN GOTO HandleError
      bsd += 1
      RregtoQreg()
      bsd += 1
      IF NOT TopOfStack() THEN GOTO HandleError
      bsd += 1
      IF NOT PopStack() THEN GOTO HandleError
      bsd += 1
      RregtoPreg()
      bsd += 1
      IF NOT Cdr() THEN GOTO HandleError
```

```
      ptype = qtype
      pvalue = qvalue
      bsd += 1
      RregtoQreg()
      bsd += 1
      IF NOT Cdr() THEN GOTO HandleError
      ptype = qtype
      pvalue = qvalue
      bsd += 1
      RregtoQreg()
      GOTO EqualP
    ELSE
      bsd -= 1
      RETURN
    END IF
  END IF

Reverse:
  dtype = ptype
  dvalue = pvalue
  etype = naught
  evalue = FALSE
ReverseLoop:
  IF dtype = naught THEN
    rtype = etype
    rvalue = evalue
    bsd -= 1
    RETURN
  END IF
  ptype = dtype
  pvalue = dvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  qtype = etype
  qvalue = evalue
  bsd += 1
  GOSUB Cons
  etype = rtype
  evalue = rvalue
  ptype = dtype
  pvalue = dvalue
```

```
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  dtype = rtype
  dvalue = rvalue
  GOSUB ReverseLoop
  RETURN

Boundp:
  ctype = ptype
  cvalue = pvalue
  tenv = curenv
  ptype = pair
  pvalue = tenv
FindBindingLoop:
  IF ptype = naught THEN
    truthtest = FALSE
    bsd -= 1
    RETURN
  END IF
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  GOSUB Assq
  IF rtype <> naught THEN
    truthtest = TRUE
    bsd -= 1
    RETURN
  END IF
  ptype = pair
  pvalue = tenv
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  tenv = pvalue
  GOSUB FindBindingLoop
  RETURN

GetFunArgs:
```

```
  ctype = ptype
  cvalue = pvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = pair
  pvalue = curenv
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  GOSUB LispEval
  ctype = rtype
  cvalue = rvalue
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  curenv = rvalue
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  ptype = naught
  pvalue = FALSE
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
GetArgValuesLoop:
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = pair
```

```
  pvalue = curenv
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  GOSUB LispEval
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  curenv = rvalue
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  ctype = rtype
  cvalue = rvalue
  dtype = qtype
  dvalue = qvalue
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
```

```
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  bsd += 1
  GOSUB Cons
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  qtype = dtype
  qvalue = dvalue
  IF qtype = naught THEN GOTO ExitGetArgLoop
  GOTO GetArgValuesLoop
ExitGetArgLoop:
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  ptype = ctype
  pvalue = cvalue
  bsd -= 1
  RETURN

CheckDot:
  ipos+=1
  IF ipos > LEN(ibuf) THEN
    PRINT "ERROR: Read.\n"
    GOTO HandleError
  END IF
  curchar=MID(ibuf, ipos, 1)
  IF curchar >= "0" AND curchar <= "9" THEN
    decimalp = TRUE
    decimal += 1
    GOTO GetNumber
  END IF
  IF curchar = " " THEN
    ctype = actsym
    cvalue = dot
    bsd -= 1
    RETURN
  END IF
  ipos = opos
  GetSymbol()
```

```
  RETURN

CheckSign:
  IF ipos + 1 > LEN(ibuf) THEN
    GetSymbol()
    RETURN
  END IF
  IF curchar = "-" THEN numsign = -1
  ipos += 1
  curchar = MID(ibuf, ipos, 1)
  IF curchar = "." THEN GOTO CheckDot
  IF curchar >= "0" AND curchar <= "9" THEN GOTO GetNumber
  ipos = opos
  GetSymbol()
  RETURN

GetNumber:
  total = VAL(curchar)
BuildNumberLoop:
  IF decimalp THEN decimal += 1
  ipos += 1
  IF ipos > LEN(ibuf) THEN
    MakeNumber()
    RETURN
  END IF
  curchar = MID(ibuf, ipos, 1)
  IF curchar = "." THEN
    IF NOT decimalp THEN
      decimalp = TRUE
      GOTO BuildNumberLoop
    ELSE
      ipos = opos
      GetSymbol()
      RETURN
    END IF
  END IF
  IF curchar >= "0" AND curchar <= "9" THEN
    total = total * 10 + VAL(curchar)
    GOTO BuildNumberLoop
  END IF
  IF INSTR("()'", curchar) <> undef OR curchar <= " " THEN
     MakeNumber()
     RETURN
```

```
  END IF
  ipos = opos
  GetSymbol()
  RETURN

Length:
  qtype = number
  qvalue = 0
LengthLoop:
  IF ptype = naught THEN
    rtype = qtype
    rvalue = qvalue
    bsd -= 1
    RETURN
  END IF
  qvalue += 1
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  GOSUB LengthLoop
  RETURN

Memq:
  ctype = ptype
  cvalue = pvalue
  ptype = qtype
  pvalue = qvalue
MemqLoop:
  IF ptype = naught THEN
    rtype = naught
    rvalue = FALSE
    bsd -= 1
    RETURN
  END IF
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  IF rtype = ctype AND rvalue = cvalue THEN
    rtype = ptype
    rvalue = pvalue
    bsd -= 1
    RETURN
  END IF
```

```
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  GOTO MemqLoop

Member:
  dtype = ptype
  dvalue = pvalue
  etype = qtype
  evalue = qvalue
MemberLoop:
  IF etype = naught THEN
    rtype = naught
    rvalue = FALSE
    bsd -= 1
    RETURN
  END IF
  ptype = etype
  pvalue = evalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  ptype = dtype
  pvalue = dvalue
  bsd += 1
  GOSUB EqualP
  IF rvalue THEN
    rtype = etype
    rvalue = evalue
    bsd -= 1
    RETURN
  END IF
  ptype = etype
  pvalue = evalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  etype = rtype
  evalue = rvalue
  GOTO MemberLoop

Assoc:
```

```
  dtype = ptype
  dvalue = pvalue
  etype = qtype
  evalue = qvalue
AssocLoop:
  IF etype = naught THEN
    rtype = naught
    rvalue = FALSE
    bsd -= 1
    RETURN
  END IF
  ptype = etype
  pvalue = evalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  qtype = dtype
  qvalue = dvalue
  bsd += 1
  GOSUB EqualP
  IF rvalue THEN
    ptype = etype
    pvalue = evalue
    IF NOT Car() THEN GOTO HandleError
    RETURN
  END IF
  ptype = etype
  pvalue = evalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  etype = rtype
  evalue = rvalue
  GOTO AssocLoop

GetCdrList:
  done = FALSE
  bsd += 1
  GOSUB Reverse
```

```
  dtype = rtype
  dvalue = rvalue
  etype = naught
  evalue = FALSE
  WHILE dtype<>naught
    ptype = dtype
    pvalue = dvalue
    bsd += 1
    IF NOT Car() THEN GOTO HandleError
    bsd += 1
    RregtoPreg()
    bsd += 1
    IF NOT Cdr() THEN GOTO HandleError
    bsd += 1
    RregtoPreg()
    IF ptype = naught THEN done = TRUE
    qtype = etype
    qvalue = evalue
    bsd += 1
    GOSUB Cons
    etype = rtype
    evalue = rvalue
    ptype = dtype
    pvalue = dvalue
    bsd += 1
    IF NOT Cdr() THEN GOTO HandleError
    dtype = rtype
    dvalue = rvalue
  WEND
  rtype = etype
  rvalue = evalue
  bsd -= 1
  RETURN

ForEach:
  btype = ptype
  bvalue = pvalue
  ctype = qtype
  cvalue = qvalue
  done = FALSE
  ForEachLoop:
  IF done THEN
    rtype = naught
```

```
    rvalue = FALSE
    bsd -= 1
    RETURN
  END IF
  ptype = btype
  pvalue = bvalue
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  bsd += 1
  GOSUB GetCarList
  bsd += 1
  RregtoQreg()
  ptype = btype
  pvalue = bvalue
  bsd += 1
  GOSUB LispApply
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  ctype = rtype
  cvalue = rvalue
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  btype = rtype
  bvalue = rvalue
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  GOSUB GetCdrList
  ctype = rtype
  cvalue = rvalue
  GOSUB ForEachLoop
  RETURN

BindParameters:
  ctype = ptype
```

```
  cvalue = pvalue
  dtype = heaptypeactive[cvalue]
  dvalue = heapvalueactive[cvalue]
  etype = qtype
  evalue = qvalue
  btype = naught
  bvalue = FALSE
BindVariablesLoop:
  IF dtype = naught THEN
    GOSUB ExtendEnvironment
    RETURN
  END IF
  IF etype = naught THEN
    PRINT "ERROR: Too few args.\n"
    GOTO HandleError
  END IF
  ptype = etype
  pvalue = evalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  ptype = dtype
  pvalue = dvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  IF ptype <> symbol THEN
    PRINT "ERROR: Variable must be symbol.\n"
    GOTO HandleError
  END IF
  bsd += 1
  GOSUB Cons
  bsd += 1
  RregtoPreg()
  qtype = btype
  qvalue = bvalue
  bsd += 1
  GOSUB Cons
  btype = rtype
  bvalue = rvalue
  ptype = dtype
```

```
  pvalue = dvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  dtype = rtype
  dvalue = rvalue
  ptype = etype
  pvalue = evalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  etype = rtype
  evalue = rvalue
  GOSUB BindVariablesLoop
  RETURN

EvalCombination:
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  IF rtype = symbol AND rvalue = scmquote THEN
    bsd += 1
    IF NOT Cdr() THEN GOTO HandleError
    bsd += 1
    RregtoPreg()
    IF NOT Car() THEN GOTO HandleError
    RETURN
  END IF
  IF rtype = symbol AND rvalue = scmdef THEN
    GOSUB DoDefine
    RETURN
  END IF
  IF rtype = symbol AND rvalue = scmset THEN
    GOSUB DoSet
    RETURN
  END IF
  IF rtype = symbol AND rvalue = scmlambda THEN
    GOSUB DoLambda
    RETURN
  END IF
  IF rtype = symbol AND rvalue = scmif THEN
    GOSUB DoIf
    RETURN
  END IF
  IF rtype = symbol AND rvalue = scmcond THEN
    GOSUB DoCond
```

```
    RETURN
  END IF
  IF rtype = symbol AND rvalue = scmseq THEN
    bsd += 1
    IF NOT Cdr() THEN GOTO HandleError
    bsd += 1
    RregtoPreg()
    GOSUB EvalSequence
    RETURN
  END IF
  IF rtype = symbol AND rvalue = scmand THEN
    GOSUB DoAnd
    RETURN
  END IF
  IF rtype = symbol AND rvalue = scmor THEN
    GOSUB DoOr
    RETURN
  END IF
  IF rtype = symbol AND rvalue = scmlet THEN
    GOSUB DoLet
    RETURN
  END IF
  IF rtype = symbol AND rvalue = scmletstar THEN
    GOSUB DoLetStar
    RETURN
  END IF
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  IF rtype = naught THEN
    GOSUB EvalNoArg
    RETURN
  END IF
  GOSUB EvalFunction
  RETURN

EvalSymbol:
  ctype = ptype
  cvalue = pvalue
  tenv = curenv
SearchFramesLoop:
  IF ptype = naught THEN
    PRINT "ERROR: Unbound variable.\n"
    GOTO HandleError
```

```
  END IF
  ptype = pair
  pvalue = tenv
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  GOSUB Assq
  IF rtype <> naught THEN
    bsd += 1
    RregtoPreg()
    IF NOT Cdr() THEN GOTO HandleError
    RETURN
  END IF
  ptype = pair
  pvalue = tenv
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  tenv = pvalue
  GOTO SearchFramesLoop

EvalFunction:
  bsd += 1
  GOSUB GetFunArgs
  bsd += 1
  RregtoQreg()
  ptype = ctype
  pvalue = cvalue
  GOSUB LispApply
  RETURN

EvalNoArg:
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  ptype = pair
  pvalue = curenv
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
```

```
  bsd += 1
  RregtoPreg()
  bsd += 1
  GOSUB LispEval
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  curenv = rvalue
  qtype = naught
  qvalue = FALSE
  GOSUB LispApply
  RETURN

LispApplyPrimitive:
  IF pvalue = 1 THEN
    GOSUB DoCons
    RETURN
  END IF
  IF pvalue = 2 THEN
    GOSUB DoCar
    RETURN
  END IF
  IF pvalue = 3 THEN
    GOSUB DoCdr
    RETURN
  END IF
  IF pvalue = 4 THEN
    GOSUB DoEq
    RETURN
  END IF
  IF pvalue = 5 THEN
    GOSUB DoNullp
    RETURN
  END IF
  IF pvalue = 6 THEN
    GOSUB DoNumEql
    RETURN
  END IF
  IF pvalue = 7 THEN
    GOSUB DoAdd
```

```
    RETURN
  END IF
  IF pvalue = 8 THEN
    GOSUB DoSub
    RETURN
  END IF
  IF pvalue = 9 THEN
    GOSUB DoMul
    RETURN
  END IF
  IF pvalue = 10 THEN
    GOSUB DoDiv
    RETURN
  END IF
  IF pvalue = 11 THEN
    GOSUB DoQit
    RETURN
  END IF
  IF pvalue = 12 THEN
    GOSUB DoNump
    RETURN
  END IF
  IF pvalue = 13 THEN
    GOSUB DoSymp
    RETURN
  END IF
  IF pvalue = 14 THEN
    GOSUB DoPrp
    RETURN
  END IF
  IF pvalue = 15 THEN
    GOSUB DoPrcp
    RETURN
  END IF
  IF pvalue = 16 THEN
    GOSUB DoBndp
    RETURN
  END IF
  IF pvalue = 17 THEN
    GOSUB DoAsq
    RETURN
  END IF
  IF pvalue = 18 THEN
```

```
    GOSUB DoRev
    RETURN
  END IF
  IF pvalue = 19 THEN
    GOSUB DoLst
    RETURN
  END IF
  IF pvalue = 20 THEN
    GOSUB DoRd
    RETURN
  END IF
  IF pvalue = 21 THEN
    GOSUB DoPrt
    RETURN
  END IF
  IF pvalue = 22 THEN
    PRINT
    rtype = booleant
    rvalue = TRUE
    bsd -= 1
    RETURN
  END IF
  IF pvalue = 23 THEN
    GOSUB DoRpla
    RETURN
  END IF
  IF pvalue = 24 THEN
    GOSUB DoRpld
    RETURN
  END IF
  IF pvalue = 25 THEN
    GOSUB DoBgrE
    RETURN
  END IF
  IF pvalue = 26 THEN
    GOSUB DoLssE
    RETURN
  END IF
  IF pvalue = 27 THEN
    GOSUB DoBgr
    RETURN
  END IF
  IF pvalue = 28 THEN
```

```
    GOSUB DoLss
    RETURN
  END IF
  IF pvalue = 29 THEN
    GOSUB DoExp
    RETURN
  END IF
  IF pvalue = 30 THEN
    GOSUB DoLog
    RETURN
  END IF
  IF pvalue = 31 THEN
    GOSUB DoFlr
    RETURN
  END IF
  IF pvalue = 32 THEN
    GOSUB DoMmq
    RETURN
  END IF
  IF pvalue = 33 THEN
    GOSUB DoLen
    RETURN
  END IF
  IF pvalue = 34 THEN
    GOSUB DoEqTre
    RETURN
  END IF
  IF pvalue = 35 THEN
    GOSUB DoMmbr
    RETURN
  END IF
  IF pvalue = 36 THEN
    GOSUB DoAssc
    RETURN
  END IF
  IF pvalue = 37 THEN
    GOSUB DoSin
    RETURN
  END IF
  IF pvalue = 38 THEN
    GOSUB DoCos
    RETURN
  END IF
```

```
  IF pvalue = 39 THEN
    GOSUB DoAtan
    RETURN
  END IF
  IF pvalue = 40 THEN
    GOSUB DoGC
    RETURN
  END IF
  IF pvalue = 41 THEN
    GOSUB DoEvl
    RETURN
  END IF
  IF pvalue = 42 THEN
    GOSUB DoAply
    RETURN
  END IF
  IF pvalue = 43 THEN
    GOSUB DoFrEch
    RETURN
  END IF
  IF pvalue = 44 THEN
    GOSUB DoMp
    RETURN
  END IF
  IF pvalue = 45 THEN
    GOSUB DoLoad
    RETURN
  END IF

DoDefine:
  etype = ptype
  evalue = pvalue
  ptype = pair
  pvalue = curenv
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = etype
  pvalue = evalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  dtype = ptype
```

```
  dvalue = pvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  IF ptype <> symbol THEN
    PRINT "ERROR: Bad type in definition.\n"
    GOTO HandleError
  END IF
  bsd += 1
  GOSUB Boundp
  IF truthtest THEN
    PRINT "ERROR: Redefinition.\n"
    GOTO HandleError
  END IF
  ptype = dtype
  pvalue = dvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  GOSUB LispEval
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  etype = rtype
  evalue = rvalue
  bsd += 1
  GOSUB Cons
  bsd += 1
  RregtoQreg()
```

```
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd+=1
  IF NOT PopStack() THEN GOTO HandleError
  bsd+=1
  RregtoPreg()
  curenv = pvalue
  dtype = ptype
  dvalue = pvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  RregtoQreg()
  bsd += 1
  GOSUB Cons
  bsd += 1
  RregtoQreg()
  ptype = dtype
  pvalue = dvalue
  bsd += 1
  IF NOT SetCar() THEN GOTO HandleError
  rtype = etype
  rvalue = evalue
  bsd -= 1
  RETURN

DoLambda:
  ctype = ptype
  cvalue = pvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  GOSUB Reverse
  ptype = ctype
  pvalue = cvalue
```

```
  ctype = rtype
  cvalue = rvalue
  IF hpcursor + 3 > maxheapsize THEN
    bsd += 1
    GOSUB GarbageCollect
  END IF
  heaptypeactive[hpcursor] = pair
  heapvalueactive[hpcursor] = curenv
  hpcursor += 1
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  heaptypeactive[hpcursor] = rtype
  heapvalueactive[hpcursor] = rvalue
  hpcursor = hpcursor + 1
  heaptypeactive[hpcursor] = ctype
  heapvalueactive[hpcursor] = cvalue
  rtype = procedure
  rvalue = hpcursor
  hpcursor += 1
  bsd -= 1
  RETURN

DoEq:
  ctype = qtype
  cvalue = qvalue
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
```

```
  bsd += 1
  RregtoPreg()
  IF ptype = qtype AND pvalue = qvalue THEN
    rtype = booleant
    rvalue = TRUE
    bsd -= 1
    RETURN
  END IF
  rtype = naught
  rvalue = FALSE
  bsd -= 1
  RETURN

DoCar:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  IF NOT Car() THEN GOTO HandleError
  RETURN

DoCond:
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  IF ptype = naught THEN
    rtype = naught
    rvalue = FALSE
    bsd -= 1
    RETURN
  END IF
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = pair
  pvalue = curenv
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
```

```
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  IF ptype = symbol AND pvalue = scmelse THEN GOTO ExitCondLoop
  bsd += 1
  GOSUB LispEval
  IF rtype <> naught THEN GOTO ExitCondLoop
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  curenv = rvalue
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  GOTO DoCond
ExitCondLoop:
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  curenv = rvalue
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
```

```
  GOSUB EvalSequence
  RETURN

DoSet:
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  dtype = ptype
  dvalue = pvalue
  ptype = pair
  pvalue = curenv
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = dtype
  pvalue = dvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  IF ptype <> symbol THEN
    PRINT "ERROR: Bad type in SET!\n"
    GOTO HandleError
  END IF
  ptype = dtype
  pvalue = dvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  GOSUB LispEval
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
```

```
  RregtoPreg()
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  curenv = rvalue
  ctype = ptype
  cvalue = pvalue
  etype = qtype
  evalue = qvalue
  tenv = curenv
FindVariableSetLoop:
  IF ptype = naught THEN
    PRINT "ERROR: Unbound variable.\n"
    GOTO HandleError
  END IF
  ptype = pair
  pvalue = tenv
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  GOSUB Assq
  IF rtype <> naught THEN
    bsd += 1
    RregtoPreg()
    qtype = etype
    qvalue = evalue
    IF NOT SetCdr() THEN GOTO HandleError
    RETURN
  END IF
  ptype = pair
  pvalue = tenv
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  tenv = pvalue
```

```
  GOTO FindVariableSetLoop

DoIf:
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  ctype = rtype
  cvalue = rvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  dtype = rtype
  dvalue = rvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  ptype = symbol
  pvalue = scmelse
  bsd += 1
  GOSUB Cons
  bsd += 1
  RregtoPreg()
  qtype = naught
  qvalue = FALSE
  bsd += 1
  GOSUB Cons
  etype = rtype
  evalue = rvalue
  ptype = dtype
  pvalue = dvalue
  qtype = naught
  qvalue = FALSE
  bsd += 1
  GOSUB Cons
  bsd += 1
  RregtoQreg()
```

```
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  GOSUB Cons
  bsd += 1
  RregtoPreg()
  qtype = etype
  qvalue = evalue
  bsd += 1
  GOSUB Cons
  bsd += 1
  RregtoQreg()
  ptype = symbol
  pvalue = scmcond
  bsd += 1
  GOSUB Cons
  bsd += 1
  RregtoPreg()
  GOSUB LispEval
  RETURN

DoAnd:
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  IF ptype = naught THEN
    rtype = booleant
    rvalue = TRUE
    bsd -= 1
    RETURN
  END IF
AndLoop:
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  IF rtype = naught THEN
    ptype = qtype
```

```
    pvalue = qvalue
    GOSUB LispEval
    RETURN
  END IF
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = pair
  pvalue = curenv
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = qtype
  pvalue = qvalue
  GOSUB LispEval
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  curenv=rvalue
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  IF qtype = naught THEN
    rtype = naught
    rvalue = FALSE
    bsd -= 1
    RETURN
  END IF
  GOTO AndLoop

DoOr:
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
OrLoop:
  IF ptype = naught THEN
    rtype = naught
    rvalue = FALSE
```

```
    bsd -= 1
    RETURN
  END IF
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = pair
  pvalue = curenv
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = qtype
  pvalue = qvalue
  GOSUB LispEval
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  curenv = rvalue
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  IF qtype <> naught THEN
    rtype = qtype
    rvalue = qvalue
    bsd -= 1
    RETURN
  END IF
  GOTO OrLoop

DoLetStar:
  bsd += 1
```

```
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  btype = rtype
  bvalue = rvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  ctype = rtype
  cvalue = rvalue
  ptype = btype
  pvalue = bvalue
  bsd += 1
  GOSUB Reverse
  btype = rtype
  bvalue = rvalue
  ptype = btype
  pvalue = bvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  qtype = naught
  qvalue = FALSE
  bsd += 1
  GOSUB Cons
  bsd += 1
  RregtoPreg()
  qtype = ctype
  qvalue = cvalue
  bsd += 1
  GOSUB Cons
  bsd += 1
  RregtoQreg()
  ptype = symbol
  pvalue = scmlet
  bsd += 1
  GOSUB Cons
  bsd += 1
  RregtoQreg()
  ptype = btype
  pvalue = bvalue
```

```
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  btype = rtype
  bvalue = rvalue
  ptype = qtype
  pvalue = qvalue
  WHILE btype <> naught
    dtype = ptype
    dvalue = pvalue
    ptype = btype
    pvalue = bvalue
    bsd += 1
    IF NOT Car() THEN GOTO HandleError
    bsd += 1
    RregtoPreg()
    qtype = naught
    qvalue = FALSE
    bsd += 1
    GOSUB Cons
    bsd += 1
    RregtoQreg()
    ptype = dtype
    pvalue = dvalue
    dtype = qtype
    dvalue = qvalue
    qtype = naught
    qvalue = FALSE
    bsd += 1
    GOSUB Cons
    bsd += 1
    RregtoQreg()
    ptype = dtype
    pvalue = dvalue
    bsd += 1
    GOSUB Cons
    bsd += 1
    RregtoQreg()
    ptype = symbol
    pvalue = scmlet
    bsd += 1
    GOSUB Cons
    bsd += 1
    RregtoQreg()
```

```
    ptype = btype
    pvalue = bvalue
    bsd += 1
    IF NOT Cdr() THEN GOTO HandleError
    btype = rtype
    bvalue = rvalue
    ptype = qtype
    pvalue = qvalue
  WEND
  GOSUB LispEval
  RETURN

DoLet:
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  btype = rtype
  bvalue = rvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  ctype = rtype
  cvalue = rvalue
  ptype = btype
  pvalue = bvalue
  bsd += 1
  GOSUB Reverse
  btype = rtype
  bvalue = rvalue
  dtype = naught
  dvalue = FALSE
  etype = naught
  evalue = FALSE
  WHILE btype <> naught
    ptype = btype
    pvalue = bvalue
    bsd += 1
    IF NOT Car() THEN GOTO HandleError
    bsd += 1
    RregtoPreg()
    bsd += 1
```

```
    IF NOT Car() THEN GOTO HandleError
    bsd += 1
    RregtoPreg()
    qtype = dtype
    qvalue = dvalue
    bsd += 1
    GOSUB Cons
    dtype = rtype
    dvalue = rvalue
    ptype = btype
    pvalue = bvalue
    bsd += 1
    IF NOT Car() THEN GOTO HandleError
    bsd += 1
    RregtoPreg()
    bsd += 1
    IF NOT Cdr() THEN GOTO HandleError
    bsd += 1
    RregtoPreg()
    bsd += 1
    IF NOT Car() THEN GOTO HandleError
    bsd += 1
    RregtoPreg()
    qtype = etype
    qvalue = evalue
    bsd += 1
    GOSUB Cons
    etype = rtype
    evalue = rvalue
    ptype = btype
    pvalue = bvalue
    bsd += 1
    IF NOT Cdr() THEN GOTO HandleError
    btype = rtype
    bvalue = rvalue
  WEND
  ptype = dtype
  pvalue = dvalue
  qtype = ctype
  qvalue = cvalue
  bsd += 1
  GOSUB Cons
  bsd += 1
```

```
  RregtoQreg()
  ptype = symbol
  pvalue = scmlambda
  bsd += 1
  GOSUB Cons
  bsd += 1
  RregtoPreg()
  qtype = etype
  qvalue = evalue
  bsd += 1
  GOSUB Cons
  bsd += 1
  RregtoPreg()
  GOSUB LispEval
  RETURN

DoCons:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  GOSUB Cons
  RETURN

DoCdr:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  IF NOT Cdr() THEN GOTO HandleError
  RETURN
```

```
DoNullp:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  IF ptype = naught THEN
    rtype = booleant
    rvalue = TRUE
    bsd -= 1
    RETURN
  END IF
  rtype = naught
  rvalue = FALSE
  bsd -= 1
  RETURN

DoNumEql:
  GOSUB DoEq
  RETURN

DoAdd:
  ptype = qtype
  pvalue = qvalue
  total = 0
SummationLoop:
  IF ptype = naught THEN
    rtype = number
    rvalue = total
    bsd -= 1
    RETURN
  END IF
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  IF rtype <> number THEN
    PRINT "ERROR: In +\n"
    GOTO HandleError
  END IF
  total += rvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
```

```
  bsd += 1
  RregtoPreg()
  GOTO SummationLoop

DoDiv:
  ptype = qtype
  pvalue = qvalue
  total = 1
  last = 0
DivLoop:
  IF ptype = naught THEN
    rtype = number
    rvalue = last / (total / last)
    bsd -= 1
    RETURN
  END IF
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  IF rtype <> number OR rvalue = 0 THEN
    PRINT "ERROR: In /\n"
    GOTO HandleError
  END IF
  last = rvalue
  total *= rvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  GOTO DivLoop

DoMul:
  ptype = qtype
  pvalue = qvalue
  total = 1
  iters = 0
TimesLoop:
  IF ptype = naught THEN
    rtype = number
    IF iters THEN
      rvalue = total
    ELSE
      rvalue = 0
    END IF
```

```
    bsd -= 1
    RETURN
  END IF
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  IF rtype <> number THEN
    PRINT "ERROR: In *\n"
    GOTO HandleError
  END IF
  total *= rvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  iters += 1
  GOTO TimesLoop

DoSub:
  ptype = qtype
  pvalue = qvalue
  total = 0
  last = 0
  iters = 0
SubtractionLoop:
  IF ptype = naught THEN
    rtype = number
    IF iters = 1 THEN
      rvalue = total - last
    ELSE
      rvalue = last - total
    END IF
    bsd -= 1
    RETURN
  END IF
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  IF rtype <> number THEN
    PRINT "ERROR: In -\n"
    GOTO HandleError
  END IF
  last = 2 * rvalue
  total += rvalue
  bsd += 1
```

```
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  iters += 1
  GOTO SubtractionLoop

DoNump:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  IF rtype = number THEN
    rtype = booleant
    rvalue = TRUE
  ELSE
    rtype = naught
    rvalue = FALSE
  END IF
  bsd -= 1
  RETURN

DoSymp:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  IF rtype = symbol THEN
    rtype = booleant
    rvalue = TRUE
  ELSE
    rtype = naught
    rvalue = FALSE
  END IF
  bsd -= 1
  RETURN

DoPrp:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  IF rtype = pair THEN
    rtype = booleant
```

```
    rvalue = TRUE
  ELSE
    rtype = naught
    rvalue = FALSE
  END IF
  bsd -= 1
  RETURN

DoPrcp:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  IF rtype = procedure OR rtype = primitive THEN
    rtype = booleant
    rvalue = TRUE
  ELSE
    rtype = naught
    rvalue = FALSE
  END IF
  bsd -= 1
  RETURN

DoBndp:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  GOSUB Boundp
  IF truthtest THEN
    rtype = booleant
    rvalue = TRUE
  ELSE
    rtype = naught
    rvalue = FALSE
  END IF
  bsd -= 1
  RETURN

DoAsq:
```

```
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  GOSUB Assq
  RETURN

DoRev:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  GOSUB Reverse
  RETURN

DoLst:
  ptype = qtype
  pvalue = qvalue
  GOSUB Reverse
  RETURN

DoRd:
  GOSUB LispRead
  RETURN

DoPrt:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
```

```
  RregtoPreg()
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  bsd += 1
  GOSUB LispPrint
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  IF NOT PopStack() THEN GOTO HandleError
  RETURN

DoRpla:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  IF NOT SetCar() THEN GOTO HandleError
  RETURN

DoRpld:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
```

```
  IF NOT SetCdr() THEN GOTO HandleError
  RETURN

DoBgrE:
  ctype = qtype
  cvalue = qvalue
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  IF ptype <> number OR qtype <> number THEN
    PRINT "ERROR: In >=\n"
    GOTO HandleError
  END IF
  IF pvalue <= qvalue THEN
    rtype = booleant
    rvalue = TRUE
    bsd -= 1
    RETURN
  END IF
  rtype = naught
  rvalue = FALSE
  bsd -= 1
  RETURN

DoLssE:
  ctype = qtype
  cvalue = qvalue
  ptype = ctype
  pvalue = cvalue
  bsd += 1
```

```
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  IF ptype <> number OR qtype <> number THEN
    PRINT "ERROR: In <=\n"
    GOTO HandleError
  END IF
  IF pvalue >= qvalue THEN
    rtype = booleant
    rvalue = TRUE
    bsd -= 1
    RETURN
  END IF
  rtype = naught
  rvalue = FALSE
  bsd -= 1
  RETURN

DoBgr:
  ctype = qtype
  cvalue = qvalue
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  ptype = ctype
  pvalue = cvalue
```

```
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  IF ptype <> number OR qtype <> number THEN
    PRINT "ERROR: In >\n"
    GOTO HandleError
  END IF
  IF pvalue < qvalue THEN
    rtype = booleant
    rvalue = TRUE
    bsd -= 1
    RETURN
  END IF
  rtype=naught
  rvalue = FALSE
  bsd -= 1
  RETURN

DoLss:
  ctype = qtype
  cvalue = qvalue
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  IF ptype <> number OR qtype <> number THEN
    PRINT "ERROR: In <\n"
    GOTO HandleError
  END IF
  IF pvalue > qvalue THEN
```

```
    rtype = booleant
    rvalue = true
    bsd -= 1
    RETURN
  END IF
  rtype = naught
  rvalue = FALSE
  bsd -= 1
  RETURN

DoExp:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  IF rtype = number THEN
    rvalue = EXP(rvalue)
    bsd -= 1
    RETURN
  END IF
  PRINT "ERROR: In EXP\n"
  GOTO HandleError

DoLog:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  IF rtype = number THEN
    rvalue = LOG(rvalue)
    bsd -= 1
    RETURN
  END IF
  PRINT "ERROR: In LOG\n"
  GOTO HandleError

DoFlr:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  IF rtype = number THEN
    rvalue = INT(rvalue)
```

```
    bsd -= 1
    RETURN
  END IF
  PRINT "ERROR: In FLOOR\n"
  GOTO HandleError

DoMmq:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  GOSUB Memq
  RETURN

DoLen:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  GOSUB Length
  RETURN

DoEqTre:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
```

```
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  GOSUB EqualP
  RETURN

DoMmbr:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  GOSUB Member
  RETURN

DoAssc:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  GOSUB Assoc
```

```
  RETURN

DoSin:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  IF rtype = number THEN
    rvalue = SIN(rvalue)
    bsd -= 1
    RETURN
  END IF
  PRINT "ERROR: In SIN\n"
  GOTO HandleError

DoCos:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  IF rtype = number THEN
    rvalue = COS(rvalue)
    bsd -= 1
    RETURN
  END IF
  PRINT "ERROR: In COS\n"
  GOTO HandleError

DoAtan:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  IF rtype = number THEN
    rvalue = ATAN(rvalue)
    bsd -= 1
    RETURN
  END IF
  PRINT "ERROR: In ATAN\n"
  GOTO HandleError

DoGC:
  GOSUB GarbageCollect
```

```
  rtype = number
  rvalue = maxheapsize - hpcursor
  bsd -= 1
  RETURN

DoEvl:
  ptype = pair
  pvalue = curenv
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  GOSUB LispEval
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  curenv = rvalue
  rtype = qtype
  rvalue = qvalue
  bsd -= 1
  RETURN

DoAply:
  ptype = qtype
  pvalue = qvalue
  btype = ptype
  bvalue = pvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  GOSUB Reverse
  bsd += 1
  RregtoQreg()
```

```
  ptype = btype
  pvalue = bvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  GOSUB LispApply
  RETURN

DoFrEch:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  GOSUB Reverse
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoQreg()
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  RregtoQreg()
  GOSUB ForEach
  RETURN

DoMp:
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  GOSUB Reverse
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
```

```
  RregtoQreg()
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  ptype = qtype
  pvalue = qvalue
  bsd += 1
  RregtoQreg()
  GOSUB MapCar
  RETURN

ExtendEnvironment:
  ptype = btype
  pvalue = bvalue
  qtype = heaptypeactive[cvalue - 2]
  qvalue = heapvalueactive[cvalue - 2]
  bsd += 1
  GOSUB Cons
  curenv = rvalue
  ptype = heaptypeactive[cvalue - 1]
  pvalue = heapvalueactive[cvalue - 1]
EvalSequence:
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  IF rtype = naught THEN
    bsd += 1
    IF NOT Car() THEN GOTO HandleError
    bsd += 1
    RregtoPreg()
    GOSUB LispEval
    RETURN
  END IF
  ctype = ptype
  cvalue = pvalue
  ptype = pair
  pvalue = curenv
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  IF NOT Cdr() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
```

```
  bsd += 1
  IF NOT PushStack() THEN GOTO HandleError
  ptype = ctype
  pvalue = cvalue
  bsd += 1
  IF NOT Car() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  GOSUB LispEval
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  RregtoPreg()
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  bsd += 1
  IF NOT TopOfStack() THEN GOTO HandleError
  bsd += 1
  IF NOT PopStack() THEN GOTO HandleError
  curenv = rvalue
  GOSUB EvalSequence
  RETURN

LispApply:
  IF ptype = primitive THEN
    GOSUB LispApplyPrimitive
    RETURN
  END IF
  IF ptype = procedure THEN
    GOSUB BindParameters
    RETURN
  END IF
  PRINT "ERROR: Unknown application.\n"
  GOTO HandleError
```