```
File created:   10-Apr-87 18:07:52  {ERIS}<LISPUSERS>LYRIC>MESATOLISP.;4

  changes to:   (VARS MESATOLISPCOMS)
                (FNS SCAN.START SCAN.TOKEN PARSE.CEDAR PARSE.BIN PARSE.FILE)

previous date:   9-Apr-87 12:25:12  {ERIS}<LISPUSERS>LYRIC>MESATOLISP.;3

  Read Table:   INTERLISP

     Package:   INTERLISP

      Format:    XCCS
```

```
(RPAQQ MESATOLISPCOMS
       [;; MESATOLISP -- By Kelly Roach.  Lyricized by L. Masinter
        (COMS

;;; SCAN: reading mesa/cedar files

              [INITVARS (SCAN.STRING (CL:MAKE-ARRAY 256 :INITIAL-ELEMENT '#\A :ELEMENT-TYPE 'CL:CHARACTER
                                            :ADJUSTABLE T :FILL-POINTER 0))
                     (SCAN.CHAR NIL)
                     (SCAN.QDOT NIL)
                     (SCAN.BOTH.RESERVED '(! %# %( %) * + %, - %. |..| / %: ; < <= = => > >= @ ABS ALL AND ANY
                                             APPLY ARRAY BASE BEGIN BROADCAST CODE COMPUTED CONTINUE DECREASING
                                             DEFINITIONS DEPENDENT DESCRIPTOR DIRECTORY DO ELSE ENABLE END
                                             ENDCASE ENDLOOP ENTRY ERROR EXIT EXITS EXPORTS FINISHED FIRST FOR
                                             FORK FRAME FREE FROM GO GOTO IF IMPORTS IN INLINE INTERNAL ISTYPE
                                             JOIN LAST LENGTH LOCKS LONG LOOP LOOPHOLE MACHINE MAX MIN MOD
                                             MONITOR MONITORED NARROW NEW NILL NOT NOTIFY NULL OF OPEN OR ORD
                                             ORDERED OVERLAID PACKED POINTER PORT PRED PRIVATE PROC PROCEDURE
                                             PROCESS PROGRAM PUBLIC READONLY RECORD REJECT RELATIVE REPEAT
                                             RESTART RESUME RETRY RETURN RETURNS SELECT SEQUENCE SHARES SIGNAL
                                             SIZE START STATE STOP SUCC THEN THROUGH TO TRANSFER TRASH TYPE
                                             UNCOUNTED UNTIL USING VAL VAR WAIT WHILE WITH ZONE %[ %] ^ _ { %| }
                                             ~))
                     (SCAN.CEDAR.RESERVED '(CEDAR CHECKED CONS LIST PAINTED REF SAFE TRUSTED UNCHECKED UNSAFE))
                     (SCAN.MESA.RESERVED '(RESIDENT]
              (FNS SCAN.INIT SCAN.START SCAN.TEST SCAN.TESTFILE SCAN.OPENSTREAM SCAN.TOKEN SCAN.NUMBER
                   SCAN.ACCEPT SCAN.APPENDDECIMAL SCAN.APPENDOCTAL SCAN.APPENDHEX SCAN.APPENDTOSCALE
                   SCAN.VALIDFRACTION SCAN.DECIMAL SCAN.OCTAL SCAN.OCTALCHAR SCAN.HEX SCAN.FLOATING SCAN.ESCAPE)
              (P (SCAN.INIT)))
        (COMS                                                  ; PARSE *
              [INITVARS (PARSE.FILELST NIL)
                     (PARSE.STREAM NIL)
                     (PARSE.FILECOMS NIL)
                     (PARSE.LANGUAGE NIL)
                     (PARSE.DIRLST NIL)
                     (PARSE.CLASS NIL)
                     (PARSE.ATOM NIL)
                     (PARSE.CLASS2 NIL)
                     (PARSE.ATOM2 NIL)
                     (PARSE.CASEHEAD.FIRST '(WITH SELECT))
                     (PARSE.DEFHEAD.FIRST '(DEFINITIONS))
                     (PARSE.DEPENDENT.FIRST '(MACHINE))
                     (PARSE.DOTEST.FIRST '(UNTIL WHILE))
                     (PARSE.FORCLAUSE.FIRST '(FOR THROUGH))
                     (PARSE.HEAP.FIRST '(UNCOUNTED))
                     (PARSE.INTERVAL.FIRST '(%( %[))
                     (PARSE.OPTRELATION.FIRST '(%# < <= = > >= IN NOT ~))
                     (PARSE.ORDERED.FIRST '(ORDERED))
                     (PARSE.ORDERLIST.FOLLOW '(! ; END %] }))
                     (PARSE.PACKED.FIRST '(PACKED))
                     (PARSE.PREFIXOP.FIRST '(ABS BASE LENGTH LONG MAX MIN ORD PRED SUCC))
                     (PARSE.PROGHEAD.FIRST '(MONITOR PROGRAM RESIDENT))
                     (PARSE.QUALIFIER.FIRST '(%. %[ ^))
                     (PARSE.RANGE.FOLLOW '(! %) %, |..| %: ; => AND DO ELSE END ENDCASE ENDLOOP EXITS FINISHED
                                             FROM NULL OR REPEAT SELECT THEN TRASH UNTIL WHILE %] }))
                     (PARSE.TRANSFER.FIRST '(BROADCAST ERROR JOIN NOTIFY RESTART RETURN SIGNAL START TRANSFER))
                     (PARSE.TRANSFERMODE.FIRST '(ERROR PORT PROCESS PROGRAM SIGNAL))
                     (PARSE.TRANSFEROP.FIRST '(ERROR FORK JOIN NEW SIGNAL START))
                     (PARSE.TYPECONS.FIRST '(%( ARRAY BASE DESCRIPTOR ERROR FRAME LONG MACHINE MONITORED ORDERED
                                                PACKED POINTER PORT PROC PORCEDURE PROCESS PROGRAM RECORD SIGNAL
                                                UNCOUNTED VAR %[ {))
                     (PARSE.TYPEOP.FIRST '(FIRST LAST NILL))
                     (PARSE.VARIANTPART.FIRST '(PACKED SELECT SEQUENCE))
                     (PARSE.CATCHLIST.FOLLOW '(END %] }))
                     (PARSE.CONTROLID.FOLLOW '(DECREASING IN _))
                     (PARSE.DECLIST.FOLLOW '(; END }))
                     (PARSE.DEFAULTOPT.FOLLOW '(%, ; END %] }))
                     (PARSE.EXITLIST.FOLLOW '(END ENDLOOP FINISHED }))
                     (PARSE.MODULELIST.FOLLOW '(IEQP EXPORTS SHARES))
                     (PARSE.OPTARGS.FOLLOW '(; ELSE END ENDCASE ENDLOOP EXITS FINISHED REPEAT %] }))
                     (PARSE.OPTEXP.FOLLOW '(! %, ; END FROM %] }))
                     (PARSE.SCOPE.FOLLOW '(END EXITS }))
                     (PARSE.STATEMENTLIST.FOLLOW '(END ENDLOOP EXITS REPEAT }))
                     (PARSE.TYPEEXP.FOLLOW '(! %, ; = => DECREASING END EXPORTS FROM IMPORTS IN OF SHARES %] _ }
                                            ))
                     (PARSE.PREDEFINED.TYPES '(ATOM BOOL BOOLEAN CARDINAL CHAR CHARACTER CONDITION INT INTEGER
                                               MDSZone MONITORLOCK NAT REAL STRING StringBody UNSPECIFIED
                                               WORD))
                     (PARSE.RELOPS (LIST '= '%# '< '<= '> '>=))
                     (PARSE.ADDOPS (LIST '+ '-))
                     (PARSE.MULTOPS (LIST '* '/ 'MOD))
                     (PARSE.TRANSFEROPS '(SIGNAL ERROR START JOIN NEW FORK))
                     (PARSE.PREFIXOPS '(LONG ABS PRED SUCC ORD MIN MAX BASE LENGTH))
                     (PARSE.TYPEOPS '(FIRST LAST NILL))
                     (PARSE.NOTS '(~ NOT]
              (RECORDS PARSERSTATE MINTERVAL MRANGE MRELATIVE MPAINTED MENUMERATED MRECORD MVAR MARRAY
                     MDESCRIPTOR MFRAME MREF MLIST PAIRITEM DEFAULT TYPELIST TYPEITEM MPOINTER CASEHEAD BINDITEM
                     KEYITEM FIELDLIST PAIRLIST ORDERLIST KEYLIST EXPLIST)
              (FNS PARSE.MESA PARSE.CEDAR PARSE.FILE PARSE.GET.STATE PARSE.SET.STATE PARSE.BIN PARSE.VARID
                   PARSE.SMURF PARSE.THISIS.MESA PARSE.THISIS.CEDAR PARSE.MODULE PARSE.INCLUDEITEM
                   PARSE.INCLUDECHECK PARSE.SEADIRT PARSE.PROGHEAD PARSE.RESIDENT PARSE.SAFE PARSE.DEFHEAD
                   PARSE.TILDE PARSE.DEFINITIONS PARSE.DEFBODY PARSE.LOCKS PARSE.LAMBDA PARSE.MODULEITEM
                   PARSE.DECLARATION PARSE.PUBLIC PARSE.ENTRY PARSE.IDLIST PARSE.IDENTLIST PARSE.POSITION
                   PARSE.OPTBITS PARSE.INTERVAL PARSE.TYPEEXP.HERE PARSE.TYPEEXP PARSE.RANGE PARSE.TYPEAPPL
                   PARSE.TYPEAPPL.CONT PARSE.TYPEID PARSE.TYPEID.CONT PARSE.TYPECONS PARSE.TYPECONS1
                   PARSE.TYPECONS.CONT PARSE.TYPECONS.RANGE PARSE.TYPECONS.RELATIVE PARSE.TYPECONS.PAINTED
                   PARSE.TYPECONS2 PARSE.TYPECONS.INTERVAL PARSE.TYPECONS.DEPENDENT PARSE.TYPECONS.ENUMERATED
                   PARSE.TYPECONS.RECORD PARSE.TYPECONS.ORDERED PARSE.TYPECONS.VAR PARSE.TYPECONS.PACKED
                   PARSE.TYPECONS.DESCRIPTOR PARSE.TYPECONS.SAFE PARSE.TYPECONS.HEAP PARSE.TYPECONS.LONG
                   PARSE.TYPECONS.FRAME PARSE.TYPECONS.REF PARSE.TYPECONS.LIST PARSE.IDENT PARSE.ELEMENT
                   PARSE.MONITORED PARSE.DEPENDENT PARSE.RECLIST PARSE.VARIANTPAIR PARSE.PAIRITEM
                   PARSE.DEFAULTOPT PARSE.VARIANTPART PARSE.VCASEHEAD PARSE.TAGTYPE PARSE.VARIANTITEM
                   PARSE.TYPELIST PARSE.TYPEITEM PARSE.POINTERTYPE PARSE.TRANSFERMODE PARSE.INITIALIZATION
                   PARSE.INITVALUE PARSE.CHECKED PARSE.CODELIST PARSE.STATEMENT PARSE.STATEMENT1
                   PARSE.STATEMENT2 PARSE.STATEMENT.CASEHEAD PARSE.STATEMENT.FORCLAUSE PARSE.STATEMENT.RETURN
                   PARSE.STATEMENT.TRANSFER PARSE.STATEMENT.LBRACKET PARSE.STATEMENT.IF PARSE.BLOCK PARSE.SCOPE
                   PARSE.BINDITEM PARSE.EXITS PARSE.CASESTMTITEM PARSE.CASEEXPITEM PARSE.EXITITEM PARSE.CASETEST
                   PARSE.CONTROLID PARSE.FORCLAUSE PARSE.DIRECTION PARSE.DOTEST PARSE.DOEXIT PARSE.ENABLES
                   PARSE.CATCHLIST PARSE.CATCHCASE PARSE.OPTARGS PARSE.TRANSFER PARSE.KEYITEM PARSE.OPTEXP
                   PARSE.EXP PARSE.EXP1 PARSE.EXP2 PARSE.EXP.TRANSFEROP PARSE.EXP.IF PARSE.EXP.CASEHEAD
                   PARSE.EXP.LHS PARSE.EXP.LBRACKET PARSE.EXP.ERROR PARSE.EXP.DISJUNCT PARSE.DISJUNCT
                   PARSE.CONJUNCT PARSE.NEGATION PARSE.RELATION PARSE.SUM PARSE.PRODUCT PARSE.OPTRELATION
                   PARSE.RELATIONTAIL PARSE.RELOP PARSE.ADDOP PARSE.MULTOP PARSE.FACTOR PARSE.PRIMARY PARSE.ATOM
                   PARSE.PRIMARY.NIL PARSE.PRIMARY.LBRACKET PARSE.PRIMARY.PREFIXOP PARSE.PRIMARY.VAL
                   PARSE.PRIMARY.ALL PARSE.PRIMARY.NEW PARSE.PRIMARY.TYPEOP PARSE.PRIMARY.SIZE
                   PARSE.PRIMARY.ISTYPE PARSE.PRIMARY.AT PARSE.PRIMARY.DESCRIPTOR PARSE.PRIMARY.CONS
                   PARSE.PRIMARY.LIST PARSE.PRIMARY.LHS PARSE.PRIMARY.LHS.NEW PARSE.PRIMARY.LHS.CONS
                   PARSE.PRIMARY.LHS.LIST PARSE.QUALIFIER PARSE.LHS PARSE.QUALIFIER.HERE PARSE.OPTCATCH
                   PARSE.TRANSFEROP PARSE.PREFIXOP PARSE.TYPEOP PARSE.DESCLIST PARSE.DIRECTORY PARSE.IMPORTS
                   PARSE.POINTERPREFIX PARSE.EXPORTS PARSE.FIELDLIST PARSE.USING PARSE.CATCHHEAD PARSE.DECLIST
                   PARSE.PAIRLIST PARSE.VARIANTLIST PARSE.ORDERLIST PARSE.LHSLIST PARSE.INCLUDELIST
                   PARSE.MODULELIST PARSE.ELEMENTLIST PARSE.BINDLIST PARSE.STATEMENTLIST PARSE.CASESTMTLIST
                   PARSE.CASELABEL PARSE.EXITLIST PARSE.KEYLIST PARSE.CASEEXPLIST PARSE.EXPLIST PARSE.OPEN
                   PARSE.CLASS PARSE.CASEHEAD PARSE.READONLY PARSE.ORDERED PARSE.BASE PARSE.PACKED PARSE.HEAP
                   PARSE.INLINE PARSE.ARGUMENTS PARSE.INTERFACE PARSE.SHARES PARSE.DEFAULT PARSE.OPTSIZE
                   PARSE.BOUNDS PARSE.LENGTH PARSE.INDEXTYPE PARSE.ELSEPART PARSE.OTHERPART PARSE.FREE
                   PARSE.CATCHANY PARSE.NOT PARSE.NEW PARSE.OPTTYPE PARSE.ARGLIST PARSE.RETURNLIST))
        (COMS ;; BUILD

              [INITVARS (BUILD.NEXT.SCOPE NIL)
                     (BUILD.CURRENT.SCOPE NIL)
                     (BUILD.SCOPE.STACK NIL)
                     (BUILD.PREFIX NIL)
                     (BUILD.FILECOMS NIL)
                     (BUILD.BOOLEAN.FNS '(AND OR NOT type? IGREATERP ILESSP IGEQ ILEQ IEQP ZEROP MINUSP EVENP
                                          ODDP FGREATERP FLESSP FEQP GREATERP LESSP GEQ LEQ))
                     (BUILD.CARDINAL.FNS '(ADD1 CHARCODE FIX GCD IDIFFERENCE IMAX IMIN IMINUS IMOD IPLUS
                                            IQUOTIENT IREMAINDER ITIMES LOGAND LOGNOT LOGOR LOGXOR
                                            NTHCHARCODE SUB1))
                     (BUILD.MIXED.FNS '(ABS DIFFERENCE EXPT MAX MIN MINUS MOD PLUS QUOTIENT REMAINDER TIMES))
                     (BUILD.REAL.FNS '(ANTILOG ARCCOS ARCSIN ARCTAN ARCTAN2 COS FDIFFERENCE FLOAT FMAX FMIN
                                      FMINUS FMOD FPLUS FQUOTIENT FREMAINDER FTIMES LOG SIN SQRT TAN))
                     (BUILD.QUALIFY.WORDS '(FREE NEW SIZE))
                     [BUILD.CARDINAL.ARITHOP.ALIST (LIST (CONS '= 'IEQP)
                                                         (CONS '%# 'IEQP)
                                                         (CONS '< 'ILESSP)
                                                         (CONS '<= 'ILEQ)
                                                         (CONS '> 'IGREATERP)
                                                         (CONS '>= 'IGEQ)
                                                         (CONS '+ 'IPLUS)
                                                         (CONS '- 'IDIFFERENCE)
                                                         (CONS '* 'ITIMES)
                                                         (CONS '/ 'IQUOTIENT)
                                                         (CONS '0- 'IMINUS)
                                                         (CONS 'MAX 'IMAX)
                                                         (CONS 'MIN 'IMIN)
```

```
                                                   (CONS 'MOD 'IMOD]
                     [BUILD.MIXED.ARITHOP.ALIST (LIST (CONS '= 'EQP)
                                                (CONS '%# 'EQP)
                                                (CONS '< 'LESSP)
                                                (CONS '<= 'GREATERP)
                                                (CONS '> 'GREATERP)
                                                (CONS '>= 'LESSP)
                                                (CONS '+ 'PLUS)
                                                (CONS '- 'DIFFERENCE)
                                                (CONS '* 'TIMES)
                                                (CONS '/ 'QUOTIENT)
                                                (CONS '0- 'MINUS)
                                                (CONS 'MAX 'MAX)
                                                (CONS 'MIN 'MIN)
                                                (CONS 'MOD 'IMOD]
                     [BUILD.REAL.ARITHOP.ALIST (LIST (CONS '= 'FEQP)
                                               (CONS '%# 'FEQP)
                                               (CONS '< 'FLESSP)
                                               (CONS '<= 'FGREATERP)
                                               (CONS '> 'FGREATERP)
                                               (CONS '>= 'FLESSP)
                                               (CONS '+ 'FPLUS)
                                               (CONS '- 'FDIFFERENCE)
                                               (CONS '* 'FTIMES)
                                               (CONS '/ 'FQUOTIENT)
                                               (CONS '0- 'FMINUS)
                                               (CONS 'MAX 'FMAX)
                                               (CONS 'MIN 'FMIN)
                                               (CONS 'MOD 'IMOD]
                     (BUILD.CARDINAL.TYPES '(CARDINAL CHAR CHARACTER INT INTEGER NAT WORD]
              (RECORDS SCOPE)
              (FNS BUILD.INIT BUILD.PUSH.SCOPE BUILD.POP.SCOPE BUILD.GC.SCOPE BUILD.STORE.EXPORTS
                   BUILD.STORE.IDENTLIST BUILD.STORE.INTERFACES BUILD.STORE.INTERFACE BUILD.STORE.OPEN
                   BUILD.STORE.USING BUILD.INITIALIZATION BUILD.INITIALIZE.VARS BUILD.INITIALIZE.VAR
                   BUILD.INITIALIZE.FN BUILD.INITIALIZE.RECORD BUILD.RECORD BUILD.TYPE BUILD.STORE.ARGLIST
                   BUILD.STORE.RETURNLIST BUILD.STORE.PAIRLIST BUILD.STORE.PAIRITEM BUILD.STORE.VARLIST BUILD.ID
                   BUILD.FIELDID BUILD.PROCID BUILD.RECORDID BUILD.TYPEID BUILD.VARID BUILD.LOCALVARID
                   BUILD.GLOBALVARID BUILD.ULTIMATE.TYPE BUILD.REFINE.TYPE BUILD.IMMEDIATE.TYPE
                   BUILD.LOOKUP.TYPE BUILD.LOOKUP BUILD.TYPEATOM BUILD.QUALIFY BUILD.QUALIFY.PREFIXOP
                   BUILD.QUALIFY.TYPEOP BUILD.QUALIFY.EXPLIST BUILD.QUALIFY.ID BUILD.ARITH.EXP1 BUILD.ARITH.EXP2
                   BUILD.ARITH.EXP* BUILD.ARITH.ADD1SUB1 BUILD.COERCE.ARITHOP BUILD.STRONGEST.TYPE.AMONG
                   BUILD.STRONGEST.TYPE BUILD.COERCE BUILD.COERCE.MARRAY BUILD.COERCE.MLIST BUILD.COERCE.EXPLIST
                   BUILD.ALIGN BUILD.ALIGN.VALUE BUILD.ADD.TO.FILECOMS BUILD.ADD1 BUILD.CALL BUILD.CHARCODE
                   BUILD.COND BUILD.COPY.OF BUILD.FETCH BUILD.FORCLAUSE.BY BUILD.FORCLAUSE.IN
                   BUILD.FORCLAUSE.THROUGH BUILD.IN BUILD.ISTYPE BUILD.LAMBDA BUILD.NEW BUILD.OR BUILD.PROG
                   BUILD.PROGN BUILD.REPLACE BUILD.RETURN BUILD.SELECTQ BUILD.SELECTQ.FN BUILD.SELECTQ.CCLAUSE
                   BUILD.SELECTQ.TEST BUILD.SELECTQ.SCLAUSE BUILD.SELECTQ.KEY BUILD.SELECTTRUEFROM
                   BUILD.SELECTTRUEFROM.CLAUSE BUILD.SETQ BUILD.SETQ.ARRAY BUILD.SETQ.ORDERLIST BUILD.SUB1
                   BUILD.TAIL)
              (P (BUILD.INIT])
```

;; MESATOLISP -- By Kelly Roach. Lyricized by L. Masinter

;;; SCAN: reading mesa/cedar files

```
(RPAQ? SCAN.STRING (CL:MAKE-ARRAY 256 :INITIAL-ELEMENT '#\A :ELEMENT-TYPE 'CL:CHARACTER :ADJUSTABLE T
                          :FILL-POINTER 0))

(RPAQ? SCAN.CHAR NIL)

(RPAQ? SCAN.QDOT NIL)

(RPAQ? SCAN.BOTH.RESERVED
       '(! %# %( %) * + %, - %. |..| / %: ; < <= = => > >= @ ABS ALL AND ANY APPLY ARRAY BASE BEGIN BROADCAST
           CODE COMPUTED CONTINUE DECREASING DEFINITIONS DEPENDENT DESCRIPTOR DIRECTORY DO ELSE ENABLE END
           ENDCASE ENDLOOP ENTRY ERROR EXIT EXITS EXPORTS FINISHED FIRST FOR FORK FRAME FREE FROM GO GOTO IF
           IMPORTS IN INLINE INTERNAL ISTYPE JOIN LAST LENGTH LOCKS LONG LOOP LOOPHOLE MACHINE MAX MIN MOD
           MONITOR MONITORED NARROW NEW NILL NOT NOTIFY NULL OF OPEN OR ORD ORDERED OVERLAID PACKED POINTER PORT
           PRED PRIVATE PROC PROCEDURE PROCESS PROGRAM PUBLIC READONLY RECORD REJECT RELATIVE REPEAT RESTART
           RESUME RETRY RETURN RETURNS SELECT SEQUENCE SHARES SIGNAL SIZE START STATE STOP SUCC THEN THROUGH TO
           TRANSFER TRASH TYPE UNCOUNTED UNTIL USING VAL VAR WAIT WHILE WITH ZONE %[ %] ^ _ { %| } ~))

(RPAQ? SCAN.CEDAR.RESERVED '(CEDAR CHECKED CONS LIST PAINTED REF SAFE TRUSTED UNCHECKED UNSAFE))

(RPAQ? SCAN.MESA.RESERVED '(RESIDENT))

(DEFINEQ

(SCAN.INIT
  [LAMBDA NIL                                                          (* kbr%: "25-Nov-85 12:05")
    (PROG NIL
          (for ATOM in SCAN.BOTH.RESERVED do (PUTPROP ATOM 'SCAN.RESERVED 'BOTH))
          (for ATOM in SCAN.CEDAR.RESERVED do (PUTPROP ATOM 'SCAN.RESERVED 'CEDAR))
          (for ATOM in SCAN.MESA.RESERVED do (PUTPROP ATOM 'SCAN.RESERVED 'MESA])

(SCAN.START
  [LAMBDA NIL                                                    ; Edited 10-Apr-87 11:39 by Masinter
    (CL:SETF (CL:FILL-POINTER SCAN.STRING)
           0])
```

(**SCAN.TEST**

```
  [LAMBDA (STRING)                                               ; Edited  6-Apr-87 15:05 by Masinter
                                                                 (* How would scanner parse a file containing this STRING? *)
    (PROG (STREAM TOKEN)
          (SETQ STREAM (OPENSTRINGSTREAM STRING))
          (SETQ SCAN.CHAR (CL:READ-CHAR STREAM))
          (SETQ SCAN.QDOT NIL)
          (SETQ TOKEN (SCAN.TOKEN STREAM))
          (CLOSEF STREAM)
          (RETURN TOKEN])
```

(**SCAN.TESTFILE**

```
  [LAMBDA (FILE)                                                 (* kbr%: "25-Nov-85 12:05")
                                                                 (* How would scanner parse a file containing this STRING? *)
    (PROG (STREAM)
          (SETQ STREAM (SCAN.OPENSTREAM FILE))
          [do (SETQ TOKEN (SCAN.TOKEN STREAM))
              (PRINT TOKEN T)
              (COND
                 ((EQ (CAR TOKEN)
                      'EOF)
                  (RETURN]
          (CLOSEF STREAM])
```

(**SCAN.OPENSTREAM**

```
  [LAMBDA (FILE)                                                 ; Edited  6-Apr-87 15:05 by Masinter
                                                                 (* Open FILE, return STREAM. *)
    (PROG (STREAM TOKEN)
          (SETQ STREAM (OPENSTREAM FILE 'INPUT))
          (SETFILEPTR STREAM 0)
          (SETQ SCAN.CHAR (CL:READ-CHAR STREAM))
          (SETQ SCAN.QDOT NIL)
          (RETURN STREAM])
```

(**SCAN.TOKEN**

```
  [LAMBDA (STREAM)                                               ; Edited 10-Apr-87 15:59 by Masinter

    ;; Return (CLASS VALUE)

    (PROG (SCAN CLASS VALUE VALID C ADVANCE PCHAR COMMENT DASHCRLF STATE NEST)
          (CL:SETF (CL:FILL-POINTER SCAN.STRING)
                 0)
          [do (while (<= (CL:CHAR-INT SCAN.CHAR)
                         (CL:CHAR-INT '#\Space))
                 do (COND
                       ((EOFP STREAM)
                        (GO ENDFILE)))
                    (SETQ SCAN.CHAR (CL:READ-CHAR STREAM)))
              (CASE SCAN.CHAR
                  ((#\a #\b #\c #\d #\e #\f #\g #\h #\i #\j #\k #\l #\m #\n #\o #\p #\q #\r #\s #\t #\u #\v #\w
                        #\x #\y #\z)
                     (SCAN.START SCAN.CHAR)
                     [do (SCAN.ACCEPT STREAM)
                         (COND
                            ((NOT (OR (CL:ALPHA-CHAR-P SCAN.CHAR)
                                      (CL:DIGIT-CHAR-P SCAN.CHAR)))
                             (RETURN]
                     (SETQ CLASS 'ID)
                     (SETQ VALUE (MKATOM SCAN.STRING))
                     (SETQ VALID T)
                     (GO GOTNEXT))
                  ((#\A #\B #\C #\D #\E #\F #\G #\H #\I #\J #\K #\L #\M #\N #\O #\P #\Q #\R #\S #\T #\U #\V #\W
                        #\X #\Y #\Z)                             (* TBW stuff concerning HTIndex.
                                                                 *)
                     (SCAN.START SCAN.CHAR)
                     [do (SCAN.ACCEPT STREAM)
                         (COND
                            ((NOT (OR (CL:ALPHA-CHAR-P SCAN.CHAR)
                                      (CL:DIGIT-CHAR-P SCAN.CHAR)))
                             (RETURN]
                     (SETQ CLASS 'ID)
                     (SETQ VALUE (MKATOM SCAN.STRING))
                     (SETQ VALID T)
                     (GO GOTNEXT))
                  ((#\0 #\1 #\2 #\3 #\4 #\5 #\6 #\7 #\8 #\9)
                     (SCAN.START SCAN.CHAR)
                     (SETQ SCAN (SCAN.NUMBER STREAM NIL))
                     (SETQ CLASS (CAR SCAN))
                     (SETQ VALUE (CADR SCAN))
                     (SETQ VALID (CADDR SCAN))
                     (COND
                        ((NOT VALID)
                         (SCAN.ERROR)))
                     (GO GOTNEXT))
                  ((#\_ #\←)
                     (SETQ CLASS '_)
                     (SETQ VALUE CLASS)
                     (GO GETNEXT))
                  ((#\^ #\↑)
                     (SETQ CLASS '^)
                     (SETQ VALUE CLASS)
                     (GO GETNEXT))
                  ((#\, #\; #\: #\# #\+ #\* #\/ #\@ #\! #\( #\) #\[ #\] #\{ #\})
                     [SETQ CLASS (MKATOM (CHARACTER (CL:CHAR-INT SCAN.CHAR]
                     (SETQ VALUE CLASS)
                     (GO GETNEXT))
                  ((#\')
                     (SETQ SCAN.CHAR (CL:READ-CHAR STREAM))
                     (SETQ SCAN (SCAN.ESCAPE STREAM))
                     (SETQ VALUE (CAR SCAN))
                     (SETQ VALID (CADR SCAN))
                     (SETQ ADVANCE (CADDR SCAN))
                     (COND
                        ((NOT VALID)
                         (SCAN.ERROR)))
                     (SETQ CLASS 'CHAR)
                     (COND
                        (ADVANCE (GO GETNEXT))
                        (T (GO GOTNEXT))))
                  ((#\")
                     (CL:SETF (CL:FILL-POINTER SCAN.STRING)
                            0)
                     (SETQ ADVANCE T)
                     [do [COND
                            (ADVANCE (SETQ SCAN.CHAR (CL:READ-CHAR STREAM]
                         (CASE SCAN.CHAR
                             ((#\")
                                (SETQ SCAN.CHAR (\BIN STREAM))
                                (COND
                                   ((NOT (IEQP SCAN.CHAR (CHARCODE %")))
                                    (RETURN)))))
                         (SETQ SCAN (SCAN.ESCAPE STREAM))
                         (CL:VECTOR-PUSH-EXTEND (CL:INT-CHAR (CAR SCAN))
                                SCAN.STRING)
                         (SETQ VALID (CADR SCAN))
                         (SETQ ADVANCE (CADDR SCAN))
                         (COND
                            ((NOT VALID)
                             (SCAN.ERROR]
                     (SETQ VALUE (CL:COPY-SEQ SCAN.STRING))
                     [COND
                        ((OR (EQL SCAN.CHAR '#\l)
                             (EQL SCAN.CHAR '#\L))
                         (SETQ CLASS 'STRING)
                         (GO GETNEXT))
                        (T (SETQ CLASS 'STRING)
                           (COND
                              ((EQL (CL:CHAR-UPCASE SCAN.CHAR)
                                    '#\G)
                               (GO GETNEXT))
                              (T (GO GOTNEXT])
                  ((#\-)
                     (SETQ SCAN.CHAR (CL:READ-CHAR STREAM))
                     (COND
                        ((NOT (EQL SCAN.CHAR '#\-))
                         (SETQ CLASS '-)
                         (SETQ VALUE '-)
                         (GO GOTNEXT)))
                     (SETQ SCAN.CHAR '#\Null)
                     (do (SETQ PCHAR SCAN.CHAR)
                         (SETQ SCAN.CHAR (CL:READ-CHAR STREAM))
                         (CASE SCAN.CHAR
                             (#\- (COND
                                     ((EQL PCHAR '#\-)
                                      (SETQ COMMENT 'DASH)
                                      (RETURN))))
                             (#\Newline
                                (SETQ COMMENT 'CRLF)
                                (RETURN))
                             NIL))
                     (SETQ SCAN.CHAR (CL:READ-CHAR STREAM))
                     (COND
                        ((AND (EQ COMMENT 'DASH)
                              (EQL SCAN.CHAR '#\Newline))
                         (SETQ DASHCRLF T)))                    (* TBW stuff about formatting *)
                     )
                  ((#\.)
                     (COND
                        (SCAN.QDOT (SETQ SCAN.QDOT NIL)
                               (SETQ CLASS '|..|)
                               (SETQ VALUE '|..|)
                               (GO GETNEXT)))
                     (COND
                        ((EOFP STREAM)
                         (SETQ CLASS '%.)
                         (SETQ VALUE '%.)
                         (GO GOTNEXT)))
                     (SETQ SCAN.CHAR (CL:READ-CHAR STREAM))
                     (CASE SCAN.CHAR
                         (#\.
                            (SETQ CLASS '|..|)
                            (SETQ VALUE '|..|)
                            (GO GETNEXT))
                         ((#\0 #\1 #\2 #\3 #\4 #\5 #\6 #\7 #\8 #\9)
                            (SCAN.START '#\.)
                            (SETQ SCAN (SCAN.NUMBER STREAM T))
                            (SETQ CLASS (CAR SCAN))
                            (SETQ VALUE (CADR SCAN))
                            (SETQ VALID (CADDR SCAN))
                            (COND
                               ((NOT VALID)
                                (SCAN.ERROR)))
                            (GO GOTNEXT))
                         (T (SETQ CLASS '%.)
                            (SETQ VALUE '%.)
                            (GO GOTNEXT))))
                  ((#\=)
                     (SETQ SCAN.CHAR (CL:READ-CHAR STREAM))
                     (COND
                        ((EQL SCAN.CHAR '#\>)
                         (SETQ CLASS '=>)
                         (SETQ VALUE '=>)
                         (GO GETNEXT))
                        (T (SETQ CLASS '=)
                           (SETQ VALUE '=)
                           (GO GOTNEXT))))
                  ((#\<)
                     (SETQ SCAN.CHAR (CL:READ-CHAR STREAM))
                     (CASE SCAN.CHAR
                         (#\=
                            (SETQ CLASS '<=)
                            (SETQ VALUE '<=)
                            (GO GETNEXT))
                         (#\<
                            (SETQ STATE 'PLAIN)
                            (SETQ NEST 1)
                            [do (SETQ SCAN.CHAR (CL:READ-CHAR STREAM))
                                (CASE SCAN.CHAR
                                    (#\> (SELECTQ STATE
                                                  ((PLAIN LEFTBROCKET)
                                                  (SETQ STATE 'RIGHTBROCKET))
                                                  (RIGHTBROCKET (SETQ STATE 'PLAIN)
                                                                (SETQ NEST (SUB1 NEST))
                                                                (COND
                                                                   ((ZEROP NEST)
                                                                    (RETURN))))
                                                  NIL))
                                    (#\< (SELECTQ STATE
                                                  ((PLAIN RIGHTBROCKET)
                                                  (SETQ STATE 'LEFTBROCKET))
                                                  (RIGHTBROCKET (SETQ STATE 'PLAIN)
                                                                (SETQ NEST (ADD1 NEST))
                                                                (COND
                                                                   ((ZEROP NEST)
                                                                    (RETURN))))
                                                  NIL))
                                    (T (SETQ STATE 'PLAIN)))]
                            (SETQ SCAN.CHAR (CL:READ-CHAR STREAM))
                                                                (* TBW formatting stuff *)
                            )
                         (T (SETQ CLASS '<)
                            (SETQ VALUE '<)
                            (GO GOTNEXT))))
                  ((#\>)
                     (SETQ SCAN.CHAR (CL:READ-CHAR STREAM))
                     (COND
                        ((EQL SCAN.CHAR '#\=)
                         (SETQ CLASS '>=)
                         (SETQ VALUE '>=)
                         (GO GETNEXT))
                        (T (SETQ CLASS '>)
                           (SETQ VALUE '>)
                           (GO GOTNEXT))))
                  (T [SETQ CLASS (MKATOM (CHARACTER (CL:CHAR-INT SCAN.CHAR]
                    (SETQ VALUE CLASS)
                    (GO GETNEXT)))]
      GETNEXT
          (SETQ SCAN.CHAR (CL:READ-CHAR STREAM))
      GOTNEXT
          [COND
             ((EQ CLASS 'ID)
              [COND
                 ((EQ VALUE NIL)                                 (* Hack NIL to NILL because I can't put properties on NIL.
                                                                 *)
                  (SETQ VALUE 'NILL]
              (COND
                 ((GETPROP VALUE 'SCAN.RESERVED)
                  (SETQ CLASS VALUE]
          (RETURN (LIST CLASS VALUE))
      ENDFILE
          (SETQ CLASS 'EOF)
          (SETQ VALUE 'EOF)
          (RETURN (LIST CLASS VALUE])
```

(**SCAN.NUMBER**

```
  [LAMBDA (STREAM FLOAT)                                         ; Edited  6-Apr-87 15:58 by Masinter
                                                                 (* Return (CLASS VALUE VALID) *)
    (PROG (CLASS VALUE VALID HEXCOUNT HEXSIG V START SCAN)
          (SETQ HEXCOUNT 0)
          (SETQ HEXSIG 0)
          (SETQ CLASS 'LNUM)
          [do (CASE SCAN.CHAR
                  ((#\0 #\1 #\2 #\3 #\4 #\5 #\6 #\7 #\8 #\9) (SCAN.ACCEPT STREAM))
                  ((#\e #\E)
                     [SETQ HEXSIG (LOGOR HEXSIG (LLSH 1 (IDIFFERENCE (CHARCODE e)
                                                             (CHARCODE a]
                     (SETQ HEXCOUNT (ADD1 HEXCOUNT))
                     (SCAN.ACCEPT STREAM)
                     (COND
                        ([AND (IEQP HEXCOUNT 1)
                              (OR (EQL SCAN.CHAR '#\+)
                                  (EQL SCAN.CHAR '#\-]
                         (SETQ FLOAT T)
                         (SCAN.ACCEPT STREAM))))
                  ((#\a #\b #\c #\d #\e #\f)
                     [SETQ HEXSIG (LOGOR HEXSIG (LLSH 1 (IDIFFERENCE (CL:CHAR-INT SCAN.CHAR)
                                                             (CHARCODE a]
                     (SETQ HEXCOUNT (ADD1 HEXCOUNT))
                     (SCAN.ACCEPT STREAM))
                  ((#\A #\B #\C #\D #\E #\F)
                     [SETQ HEXSIG (LOGOR HEXSIG (LLSH 1 (IDIFFERENCE SCAN.CHAR (CHARCODE A]
                     (SETQ HEXCOUNT (ADD1 HEXCOUNT))
                     (SCAN.ACCEPT STREAM))
                  ((#\h #\H)
                     [SETQ HEXSIG (LOGOR HEXSIG (LLSH 1 (IDIFFERENCE (CHARCODE h)
                                                             (CHARCODE a]
                     (SETQ HEXCOUNT (ADD1 HEXCOUNT))
                     (SCAN.ACCEPT STREAM))
                  ((#\.)
                     (COND
                        ((OR (NOT (IEQP HEXCOUNT 0))
                             FLOAT)
                         (RETURN)))
                     (SETQ SCAN.CHAR (CL:READ-CHAR STREAM))
                     (COND
                        ((EQL SCAN.CHAR '#\.)
                         (SETQ SCAN.QDOT T)
                         (RETURN)))
                     (SETQ FLOAT T)
                     (CL:VECTOR-PUSH-EXTEND '#\. SCAN.STRING))
                  (T (RETURN)))]
          (CL:VECTOR-PUSH-EXTEND '#\Null SCAN.STRING)
          [COND
             (FLOAT (SETQ CLASS 'FLNUM)
                    (SETQ SCAN (SCAN.FLOATING SCAN.STRING))
                    (SETQ V (CAR SCAN))
                    (SETQ VALID (CADR SCAN)))
             ([NOT (ZEROP (LOGAND HEXSIG (LLSH 1 (IDIFFERENCE (CHARCODE h)
                                                      (CHARCODE a]
              (SETQ SCAN (SCAN.HEX SCAN.STRING))
              (SETQ V (CAR SCAN))
              (SETQ VALID (CADR SCAN)))
             ((IEQP HEXCOUNT 0)
              (SETQ SCAN (SCAN.DECIMAL SCAN.STRING))
              (SETQ V (CAR SCAN))
              (SETQ VALID (CADR SCAN)))
             ((IEQP HEXCOUNT 1)
              (SELECTC HEXSIG
                  ((LLSH 1 (IDIFFERENCE (CHARCODE b)
                                (CHARCODE a)))
```

```
                      (SETQ SCAN (SCAN.OCTAL SCAN.STRING)))
                  ((LLSH 1 (IDIFFERENCE (CHARCODE c)
                                 (CHARCODE a)))
                      (SETQ CLASS 'CHAR)
                      (SETQ SCAN (SCAN.OCTALCHAR SCAN.STRING)))
                  ((LLSH 1 (IDIFFERENCE (CHARCODE d)
                                 (CHARCODE a)))
                      (SETQ SCAN (SCAN.DECIMAL SCAN.STRING)))
                  ((LLSH 1 (IDIFFERENCE (CHARCODE e)
                                 (CHARCODE a)))
                      (SETQ CLASS 'FLNUM)
                      (SETQ SCAN (SCAN.FLOATING SCAN.STRING)))
                  (SETQ SCAN (SCAN.HEX SCAN.STRING)))
              (SETQ V (CAR SCAN))
              (SETQ VALID (CADR SCAN)))
             (T (SETQ SCAN (SCAN.HEX SCAN.STRING))
                (SETQ V (CAR SCAN))
                (SETQ VALID (CADR SCAN]                          (* TBW stuff *)
         (RETURN (LIST CLASS V VALID])
```

(SCAN.ACCEPT

```
  [LAMBDA (STREAM)                                               ; Edited  6-Apr-87 15:25 by Masinter
    (CL:VECTOR-PUSH-EXTEND SCAN.CHAR SCAN.STRING)
    (SETQ SCAN.CHAR (CL:READ-CHAR STREAM])
```

(SCAN.APPENDDECIMAL

```
  [LAMBDA (V DIGIT)                                              (* kbr%: "25-Nov-85 12:06")
                                                                 (* DIGIT is a character code. Return
                                                                 (NEWV VALID) *)

    (PROG (MAXV MAXD D VALID NEWV)
          (SETQ MAXV 429496729)
          (SETQ MAXD 5)
          (SETQ D (IDIFFERENCE DIGIT (CHARCODE 0)))
          [SETQ VALID (OR (ILESSP V MAXV)
                          (AND (IEQP V MAXV)
                               (ILEQ D MAXD]
          (SETQ NEWV (IPLUS (ITIMES 10 V)
                            D))
          (RETURN (LIST NEWV VALID])
```

(SCAN.APPENDOCTAL

```
  [LAMBDA (V DIGIT)                                              (* kbr%: "25-Nov-85 12:06")
                                                                 (* DIGIT is a character code. Return
                                                                 (NEWV VALID) *)

    (PROG (MAXV D VALID NEWV)
          (SETQ MAXV 536870911)
          (SETQ D (IDIFFERENCE DIGIT (CHARCODE 0)))
          (SETQ VALID (ILEQ V MAXV))
          (SETQ NEWV (IPLUS (ITIMES 8 V)
                            D))
          (RETURN (LIST NEWV VALID])
```

(SCAN.APPENDHEX

```
  [LAMBDA (V DIGIT)                                              (* kbr%: "25-Nov-85 12:06")
                                                                 (* DIGIT is a character code. Return
                                                                 (NEWV VALID) *)

    (PROG (MAXV D VALID NEWV)
          (SETQ MAXV 268435455)
          [COND
             [(AND (IGEQ DIGIT (CHARCODE 0))
                   (ILEQ DIGIT (CHARCODE 9)))
              (SETQ D (IDIFFERENCE DIGIT (CHARCODE 0]
             (T (SETQ D (IPLUS DIGIT (IMINUS (CHARCODE A))
                               10]
          (SETQ VALID (ILEQ V MAXV))
          (SETQ NEWV (IPLUS (ITIMES 16 V)
                            D))
          (RETURN (LIST NEWV VALID])
```

(SCAN.APPENDTOSCALE

```
  [LAMBDA (V DIGIT)                                              (* kbr%: "25-Nov-85 12:06")
                                                                 (* DIGIT is a character code. Return
                                                                 (NEWV VALID) *)

    (PROG (MAXV MAXD D VALID NEWV)
          (SETQ MAXV 6553)
          (SETQ MAXD 5)
          (SETQ D (IDIFFERENCE DIGIT (CHARCODE 0)))
          [SETQ VALID (OR (ILESSP V MAXV)
                          (AND (IEQP V MAXV)
                               (ILEQ D MAXD]
          (SETQ NEWV (IPLUS (ITIMES 10 V)
                            D))
          (RETURN (LIST NEWV VALID])

(SCAN.VALIDFRACTION
  [LAMBDA (V DIGIT)                                                    (* kbr%: "25-Nov-85 12:06")
                                                                       (* DIGIT is a character code. Return VALID.
                                                                       *)
    (PROG (MAXV MAXD D VALID)
          (SETQ MAXV 214748364)
          (SETQ MAXD 7)
          (SETQ D (IDIFFERENCE DIGIT (CHARCODE 0)))
          [SETQ VALID (OR (ILESSP V MAXV)
                          (AND (IEQP V MAXV)
                               (ILEQ D MAXD]
          (RETURN VALID])

(SCAN.DECIMAL
  [LAMBDA (BUFFER)                                                     ; Edited  6-Apr-87 15:48 by Masinter
                                                                       (* Return (VALUE VALID) *)
    (PROG (VALUE VALID BUFFERPTR C V SCAN SCALE)
          (SETQ VALID T)
          (SETQ BUFFERPTR 0)
          (SETQ V 0)
          (while (AND (IGEQ (SETQ C (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR)))
                            (CHARCODE 0))
                      (ILEQ C (CHARCODE 9)))
             do [COND
                   (VALID (SETQ SCAN (SCAN.APPENDDECIMAL V C))
                          (SETQ V (CAR SCAN))
                          (SETQ VALID (CADR SCAN]
                (SETQ BUFFERPTR (ADD1 BUFFERPTR)))
          [COND
             ((OR (IEQP C (CHARCODE d))
                  (IEQP C (CHARCODE D)))
              (SETQ SCALE 0)
              (SETQ BUFFERPTR (ADD1 BUFFERPTR))
              (while (AND (IGEQ (SETQ C (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR)))
                                (CHARCODE 0))
                          (ILEQ C (CHARCODE 9)))
                 do [COND
                       (VALID (SETQ SCAN (SCAN.APPENDTOSCALE SCALE C))
                              (SETQ SCALE (CAR SCAN))
                              (SETQ VALID (CADR SCAN]
                    (SETQ BUFFERPTR (ADD1 BUFFERPTR)))
              (for I from 1 to SCALE do (SETQ SCAN (SCAN.APPENDDECIMAL V (CHARCODE 0)))
                                        (SETQ V (CAR SCAN))
                                        (SETQ VALID (CADR SCAN]
          (COND
             ([NOT (ZEROP (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR]
              (SETQ VALID NIL)))
          (SETQ VALUE V)
          (RETURN (LIST VALUE VALID])

(SCAN.OCTAL
  [LAMBDA (BUFFER)                                                     ; Edited  6-Apr-87 15:55 by Masinter
                                                                       (* Return (VALUE VALID) *)
    (PROG (VALUE VALID BUFFERPTR C V SCAN SCALE)
          (SETQ BUFFERPTR 0)
          (SETQ VALID T)
          (SETQ V 0)
          (while (AND (IGEQ (SETQ C (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR)))
                            (CHARCODE 0))
                      (ILEQ C (CHARCODE 7)))
             do [COND
                   (VALID (SETQ SCAN (SCAN.APPENDOCTAL V C))
                          (SETQ V (CAR SCAN))
                          (SETQ VALID (CADR SCAN]
                (SETQ BUFFERPTR (ADD1 BUFFERPTR)))
          [COND
             ((OR (IEQP C (CHARCODE b))
                  (IEQP C (CHARCODE B)))
              (SETQ SCALE 0)
              (SETQ BUFFERPTR (ADD1 BUFFERPTR))
              (while (AND (IGEQ (SETQ C (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR)))
                                (CHARCODE 0))
                          (ILEQ C (CHARCODE 7)))
                 do [COND
                       (VALID (SETQ SCAN (SCAN.APPENDTOSCALE SCALE C))
                              (SETQ SCALE (CAR SCAN))
                              (SETQ VALID (CADR SCAN]
                    (SETQ BUFFERPTR (ADD1 BUFFERPTR)))
              (for I from 1 to SCALE do (SETQ SCAN (SCAN.APPENDOCTAL V (CHARCODE 0)))
                                        (SETQ V (CAR SCAN))
                                        (SETQ VALID (CADR SCAN]
          (COND
             ([NOT (ZEROP (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR]
              (SETQ VALID NIL)))
          (SETQ VALUE V)
          (RETURN (LIST VALUE VALID])
```

(**SCAN.OCTALCHAR**

```
  [LAMBDA (BUFFER)                                                  ; Edited  6-Apr-87 15:57 by Masinter
                                                                    (* Return (VALUE VALID) *)
    (PROG (VALUE VALID BUFFERPTR C V SCAN SCALE)
          (SETQ BUFFERPTR 0)
          (SETQ VALID T)
          (SETQ V 0)
          (while (AND (IGEQ (SETQ C (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR)))
                            (CHARCODE 0))
                      (ILEQ C (CHARCODE 7)))
             do [COND
                   (VALID (SETQ SCAN (SCAN.APPENDOCTAL V C))
                          (SETQ V (CAR SCAN))
                          (SETQ VALID (CADR SCAN]
                (SETQ BUFFERPTR (ADD1 BUFFERPTR)))
          [COND
             ((OR (IEQP C (CHARCODE c))
                  (IEQP C (CHARCODE C)))
              (SETQ BUFFERPTR (ADD1 BUFFERPTR]
          (COND
             ([NOT (ZEROP (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR]
              (SETQ VALID NIL)))
          (COND
             ((NOT (OR (IGEQ V 0)
                       (ILEQ V 255)))
              (SETQ VALID NIL)))
          (SETQ VALUE V)
          (RETURN (LIST VALUE VALID])
```

(**SCAN.HEX**

```
  [LAMBDA (BUFFER)                                                  ; Edited  6-Apr-87 15:45 by Masinter
                                                                    (* Return (VALUE VALID) *)
    (PROG (VALUE VALID BUFFERPTR C V SCAN SCALE)
          (SETQ BUFFERPTR 0)
          (SETQ VALID T)
          (SETQ V 0)
          (while [NOT (ZEROP (SETQ C (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR]
             do (COND
                   [[OR (AND (IGEQ C (CHARCODE 0))
                             (ILEQ C (CHARCODE 9)))
                        (AND (IGEQ C (CHARCODE A))
                             (ILEQ C (CHARCODE F]
                    (COND
                       (VALID (SETQ SCAN (SCAN.APPENDHEX V C))
                              (SETQ V (CAR SCAN))
                              (SETQ VALID (CADR SCAN]
                   [(AND (IGEQ C (CHARCODE a))
                         (ILEQ C (CHARCODE f)))
                    (COND
                       (VALID [SETQ SCAN (SCAN.APPENDHEX V (IDIFFERENCE C (IDIFFERENCE (CHARCODE a)
                                                                                    (CHARCODE A]
                              (SETQ V (CAR SCAN))
                              (SETQ VALID (CADR SCAN]
                   (T (RETURN)))
                (SETQ BUFFERPTR (ADD1 BUFFERPTR)))
          [COND
             ((OR (IEQP C (CHARCODE h))
                  (IEQP C (CHARCODE H)))
              (SETQ SCALE 0)
              (SETQ BUFFERPTR (ADD1 BUFFERPTR))
              (while (AND (IGEQ (SETQ C (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR)))
                                (CHARCODE 0))
                          (ILEQ C (CHARCODE 9)))
                 do [COND
                       (VALID (SETQ SCAN (SCAN.APPENDTOSCALE SCALE C))
                              (SETQ SCALE (CAR SCAN))
                              (SETQ VALID (CADR SCAN]
                    (SETQ BUFFERPTR (ADD1 BUFFERPTR)))
              (for I from 1 to SCALE do (SETQ SCAN (SCAN.APPENDHEX V (CHARCODE 0)))
                                        (SETQ V (CAR SCAN))
                                        (SETQ VALID (CADR SCAN]
          (COND
             ([NOT (ZEROP (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR]
              (SETQ VALID NIL)))
          (SETQ VALUE V)
          (RETURN (LIST VALUE VALID])
```

(**SCAN.FLOATING**

```
  [LAMBDA (BUFFER)                                                  ; Edited  6-Apr-87 15:46 by Masinter
```

```
                                                                          (* Return (VALUE VALID) *)
    (PROG (VALUE VALID BUFFERPTR C V EXP SCAN SCALE OP)
          (SETQ BUFFERPTR 0)
          (SETQ VALID T)
          (SETQ V 0)
          (SETQ EXP 0)
          (while (AND [<= (CHARCODE 0)
                          (SETQ C (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR]
                      (< C (CHARCODE 9)))
             do (SETQ VALID (AND VALID (SCAN.VALIDFRACTION V C)))
                [COND
                   (VALID (SETQ SCAN (SCAN.APPENDDECIMAL V C))
                          (SETQ V (CAR SCAN)))
                   (T (SETQ EXP (ADD1 EXP]
                (SETQ BUFFERPTR (ADD1 BUFFERPTR)))
          [COND
             ((= C (CHARCODE %.))
              (SETQ BUFFERPTR (ADD1 BUFFERPTR))
              (SETQ C (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR)))
              (COND
                 ([NOT (AND (IGEQ C (CHARCODE 0))
                            (ILEQ C (CHARCODE 9]
                  (SETQ VALID NIL)))
              (while (AND (IGEQ (SETQ C (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR)))
                                (CHARCODE 0))
                          (ILEQ C (CHARCODE 9)))
                 do (SETQ VALID (AND VALID (SCAN.VALIDFRACTION V C)))
                    [COND
                       (VALID (SETQ SCAN (SCAN.APPENDDECIMAL V C))
                              (SETQ V (CAR SCAN))
                              (SETQ VALID (CADR SCAN))
                              (SETQ EXP (SUB1 EXP]
                    (SETQ BUFFERPTR (ADD1 BUFFERPTR]
          (SETQ VALID T)
          [COND
             ((OR (IEQP C (CHARCODE e))
                  (IEQP C (CHARCODE E)))
              (SETQ SCALE 0)
              (SETQ OP 'PLUS)
              (SETQ BUFFERPTR (ADD1 BUFFERPTR))
              (SELCHARQ (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR))
                   ("+" (SETQ BUFFERPTR (ADD1 BUFFERPTR)))
                   ("-" (SETQ OP 'MINUS)
                        (SETQ BUFFERPTR (ADD1 BUFFERPTR)))
                   NIL)
              (COND
                 ([NOT (AND (IGEQ (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR))
                                  (CHARCODE 0))
                            (ILEQ (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR))
                                  (CHARCODE 9]
                  (SETQ VALID NIL)))
              (while (AND (IGEQ (SETQ C (CL:CHAR-INT (CL:ELT BUFFER BUFFERPTR)))
                                (CHARCODE 0))
                          (ILEQ C (CHARCODE 9)))
                 do [COND
                       (VALID (SETQ SCAN (SCAN.APPENDTOSCALE SCALE C))
                              (SETQ SCALE (CAR SCAN))
                              (SETQ VALID (CADR SCAN]
                    (SETQ BUFFERPTR (ADD1 BUFFERPTR)))
              (SETQ EXP (COND
                           ((EQ OP 'PLUS)
                            (IPLUS EXP SCALE))
                           (T (IDIFFERENCE EXP SCALE]
          (COND
             ((NOT (ZEROP (CL:ELT BUFFER BUFFERPTR)))
              (SETQ VALID NIL)))

          (* TBW NOTE%: Look at MKNUMATOM & \FLOATINGSCALE to find right way to do this.
          *)

          (SETQ VALUE (FTIMES V (EXPT 10.0 EXP)))
          (RETURN (LIST VALUE VALID])
```

(SCAN.ESCAPE

```
  [LAMBDA (STREAM)                                                        ; Edited  6-Apr-87 15:28 by Masinter
    (PROG (C VALID ADVANCE V NC)
          (SETQ VALID T)
          (SETQ ADVANCE T)
          (SETQ C SCAN.CHAR)
          [COND
             ((EQL C '#\\)
              (SETQ SCAN.CHAR (CL:READ-CHAR STREAM))
              (SETQ C (CASE SCAN.CHAR
                          ((#\n #\N #\r #\R) (CHARCODE CR))
                          ((#\l #\L) (CHARCODE LF))
                          ((#\t #\T) (CHARCODE TAB))
```

```
                             ((#\b #\B) (CHARCODE BS))
                             ((#\f #\F) (CHARCODE FF))
                             ((#\' #\" #\\) (CL:CHAR-INT SCAN.CHAR))
                             ((#\0 #\1 #\2 #\3 #\4 #\5 #\6 #\7)
                                (SETQ V 0)
                                (SETQ NC 0)
                                (do (COND
                                       ([NOT (AND (IGEQ (CL:CHAR-INT SCAN.CHAR)
                                                        (CHARCODE 0))
                                                  (ILEQ (CL:CHAR-INT SCAN.CHAR)
                                                        (CHARCODE 7]
                                        (SETQ VALID NIL)
                                        (SETQ ADVANCE NIL)
                                        (RETURN)))
                                    [SETQ V (IPLUS (ITIMES 8 V)
                                                   (IDIFFERENCE (CL:CHAR-INT SCAN.CHAR)
                                                          (CHARCODE 0]
                                    (COND
                                       ((IEQP (SETQ NC (ADD1 NC))
                                              3)
                                        (RETURN)))
                                    (SETQ SCAN.CHAR (CL:READ-CHAR STREAM)))
                                (COND
                                   ((IGREATERP V 255)
                                    (SETQ VALID NIL)
                                    (SETQ V 0)))
                                (SETQ C V))
                             (T (SETQ VALID NIL)
                                (SETQ ADVANCE NIL)))]
              (RETURN (LIST C VALID ADVANCE])

)

(SCAN.INIT)
```

;; PARSE *

```
(RPAQ? PARSE.FILELST NIL)

(RPAQ? PARSE.STREAM NIL)

(RPAQ? PARSE.FILECOMS NIL)

(RPAQ? PARSE.LANGUAGE NIL)

(RPAQ? PARSE.DIRLST NIL)

(RPAQ? PARSE.CLASS NIL)

(RPAQ? PARSE.ATOM NIL)

(RPAQ? PARSE.CLASS2 NIL)

(RPAQ? PARSE.ATOM2 NIL)

(RPAQ? PARSE.CASEHEAD.FIRST '(WITH SELECT))

(RPAQ? PARSE.DEFHEAD.FIRST '(DEFINITIONS))

(RPAQ? PARSE.DEPENDENT.FIRST '(MACHINE))

(RPAQ? PARSE.DOTEST.FIRST '(UNTIL WHILE))

(RPAQ? PARSE.FORCLAUSE.FIRST '(FOR THROUGH))

(RPAQ? PARSE.HEAP.FIRST '(UNCOUNTED))

(RPAQ? PARSE.INTERVAL.FIRST '(%( %[))

(RPAQ? PARSE.OPTRELATION.FIRST '(%# < <= = > >= IN NOT ~))

(RPAQ? PARSE.ORDERED.FIRST '(ORDERED))

(RPAQ? PARSE.ORDERLIST.FOLLOW '(! ; END %] }))

(RPAQ? PARSE.PACKED.FIRST '(PACKED))

(RPAQ? PARSE.PREFIXOP.FIRST '(ABS BASE LENGTH LONG MAX MIN ORD PRED SUCC))

(RPAQ? PARSE.PROGHEAD.FIRST '(MONITOR PROGRAM RESIDENT))

(RPAQ? PARSE.QUALIFIER.FIRST '(%. %[ ^))

(RPAQ? PARSE.RANGE.FOLLOW '(! %) %, |..| %: ; => AND DO ELSE END ENDCASE ENDLOOP EXITS FINISHED FROM NULL OR
                             REPEAT SELECT THEN TRASH UNTIL WHILE %] }))

(RPAQ? PARSE.TRANSFER.FIRST '(BROADCAST ERROR JOIN NOTIFY RESTART RETURN SIGNAL START TRANSFER))
```

```
(RPAQ? PARSE.TRANSFERMODE.FIRST '(ERROR PORT PROCESS PROGRAM SIGNAL))

(RPAQ? PARSE.TRANSFEROP.FIRST '(ERROR FORK JOIN NEW SIGNAL START))

(RPAQ? PARSE.TYPECONS.FIRST '(%( ARRAY BASE DESCRIPTOR ERROR FRAME LONG MACHINE MONITORED ORDERED PACKED
                                POINTER PORT PROC PORCEDURE PROCESS PROGRAM RECORD SIGNAL UNCOUNTED VAR %[ {))

(RPAQ? PARSE.TYPEOP.FIRST '(FIRST LAST NILL))

(RPAQ? PARSE.VARIANTPART.FIRST '(PACKED SELECT SEQUENCE))

(RPAQ? PARSE.CATCHLIST.FOLLOW '(END %] }))

(RPAQ? PARSE.CONTROLID.FOLLOW '(DECREASING IN _))

(RPAQ? PARSE.DECLIST.FOLLOW '(; END }))

(RPAQ? PARSE.DEFAULTOPT.FOLLOW '(%, ; END %] }))

(RPAQ? PARSE.EXITLIST.FOLLOW '(END ENDLOOP FINISHED }))

(RPAQ? PARSE.MODULELIST.FOLLOW '(IEQP EXPORTS SHARES))

(RPAQ? PARSE.OPTARGS.FOLLOW '(; ELSE END ENDCASE ENDLOOP EXITS FINISHED REPEAT %] }))

(RPAQ? PARSE.OPTEXP.FOLLOW '(! %, ; END FROM %] }))

(RPAQ? PARSE.SCOPE.FOLLOW '(END EXITS }))

(RPAQ? PARSE.STATEMENTLIST.FOLLOW '(END ENDLOOP EXITS REPEAT }))

(RPAQ? PARSE.TYPEEXP.FOLLOW '(! %, ; = => DECREASING END EXPORTS FROM IMPORTS IN OF SHARES %] _ }))

(RPAQ? PARSE.PREDEFINED.TYPES '(ATOM BOOL BOOLEAN CARDINAL CHAR CHARACTER CONDITION INT INTEGER MDSZone
                                MONITORLOCK NAT REAL STRING StringBody UNSPECIFIED WORD))

(RPAQ? PARSE.RELOPS (LIST '= '%# '< '<= '> '>=))

(RPAQ? PARSE.ADDOPS (LIST '+ '-))

(RPAQ? PARSE.MULTOPS (LIST '* '/ 'MOD))

(RPAQ? PARSE.TRANSFEROPS '(SIGNAL ERROR START JOIN NEW FORK))

(RPAQ? PARSE.PREFIXOPS '(LONG ABS PRED SUCC ORD MIN MAX BASE LENGTH))

(RPAQ? PARSE.TYPEOPS '(FIRST LAST NILL))

(RPAQ? PARSE.NOTS '(~ NOT))

(DECLARE%: EVAL@COMPILE

(TYPERECORD PARSERSTATE (STREAM FILEPTR CHAR QDOT CLASS ATOM CLASS2 ATOM2 PREFIX NEXTSCOPE CURRENTSCOPE
                               SCOPESTACK FILECOMS))

[TYPERECORD MINTERVAL (KIND . BOUNDS)
       (ACCESSFNS ((LBOUND (CAR (fetch (MINTERVAL BOUNDS) of DATUM)))
                   (UBOUND (CADR (fetch (MINTERVAL BOUNDS) of DATUM]

(TYPERECORD MRANGE (TYPE INTERVAL))

(TYPERECORD MRELATIVE (TYPEID TYPE))

(TYPERECORD MPAINTED (TYPEID TYPE))

(TYPERECORD MENUMERATED ITEMS)

(TYPERECORD MRECORD (RECORDID . FIELDLIST))

(TYPERECORD MVAR TYPE)

(TYPERECORD MARRAY (INDEXTYPE TYPE))

(TYPERECORD MDESCRIPTOR TYPE)

(TYPERECORD MFRAME ID)

(TYPERECORD MREF TYPE)

(TYPERECORD MLIST TYPE)

(RECORD PAIRITEM (ID TYPEEXP DEFAULT))

(RECORD DEFAULT (EXP TRASH))

(TYPERECORD TYPELIST ITEMS)
```

```
(RECORD TYPEITEM (TYPEEXP DEFAULT))

(TYPERECORD MPOINTER TYPE)

(TYPERECORD CASEHEAD (ID EXP OPTEXP))

(TYPERECORD BINDITEM (ID EXP))

(RECORD KEYITEM (ID OPTEXP))

[RECORD FIELDLIST (TYPE . ITEMS)
       (TYPE? (AND (LISTP DATUM)
                   (FMEMB (CAR DATUM)
                          '(PAIRLIST TYPELIST]

(TYPERECORD PAIRLIST ITEMS)

(TYPERECORD ORDERLIST ITEMS)

(TYPERECORD KEYLIST ITEMS)

[RECORD EXPLIST (TYPE . ITEMS)
       (TYPE? (AND (LISTP DATUM)
                   (FMEMB (CAR DATUM)
                          '(KEYLIST ORDERLIST]
)

(DEFINEQ

(PARSE.MESA
  [LAMBDA (FILE DIRLST)                                    (* kbr%: "25-Nov-85 12:46")
    (PARSE.FILE FILE 'MESA DIRLST])


(PARSE.CEDAR
  (CL:LAMBDA (&OPTIONAL FILE DIRLST)                       ; Edited 10-Apr-87 16:00 by Masinter
         (PARSE.FILE FILE 'CEDAR DIRLST)))


(PARSE.FILE
  (CL:LAMBDA (&OPTIONAL FILE LANGUAGE DIRLST)              ; Edited 10-Apr-87 16:01 by Masinter
         (PROG NIL
               (SETQ PARSE.DIRLST DIRLST)
               (SETQ PARSE.LANGUAGE LANGUAGE)
               (SETQ PARSE.STREAM (SCAN.OPENSTREAM FILE))
               (SETQ PARSE.ATOM NIL)
               (SETQ PARSE.ATOM2 NIL)
               (PARSE.BIN)
               (PARSE.BIN)
               (PARSE.MODULE)
               (SETQ PARSE.FILECOMS (DREVERSE PARSE.FILECOMS))
               (CLOSEF PARSE.STREAM))))


(PARSE.GET.STATE
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:46")
                                                           (* Get parser state to save before interruption.
                                                           *)

    (create PARSERSTATE
           STREAM _ PARSE.STREAM
           FILEPTR _ (GETFILEPTR PARSE.STREAM)
           CHAR _ SCAN.CHAR
           QDOT _ SCAN.QDOT
           CLASS _ PARSE.CLASS
           ATOM _ PARSE.ATOM
           CLASS2 _ PARSE.CLASS2
           ATOM2 _ PARSE.ATOM2
           PREFIX _ BUILD.PREFIX
           NEXTSCOPE _ BUILD.NEXT.SCOPE
           CURRENTSCOPE _ BUILD.CURRENT.SCOPE
           SCOPESTACK _ BUILD.SCOPE.STACK
           FILECOMS _ BUILD.FILECOMS])


(PARSE.SET.STATE
  [LAMBDA (STATE)                                          (* kbr%: "25-Nov-85 12:46")
                                                           (* Restore state after interruption.
                                                           *)

    (PROG NIL
          (SETQ PARSE.STREAM (fetch (PARSERSTATE STREAM) of STATE))
          (SETFILEPTR PARSE.STREAM (fetch (PARSERSTATE FILEPTR) of STATE))
          (SETQ SCAN.CHAR (fetch (PARSERSTATE CHAR) of STATE))
          (SETQ SCAN.QDOT (fetch (PARSERSTATE QDOT) of STATE))
          (SETQ PARSE.CLASS (fetch (PARSERSTATE CLASS) of STATE))
          (SETQ PARSE.ATOM (fetch (PARSERSTATE ATOM) of STATE))
          (SETQ PARSE.CLASS2 (fetch (PARSERSTATE CLASS2) of STATE))
          (SETQ PARSE.ATOM2 (fetch (PARSERSTATE ATOM2) of STATE))
          (SETQ BUILD.PREFIX (fetch (PARSERSTATE PREFIX) of STATE))
          (SETQ BUILD.NEXT.SCOPE (fetch (PARSERSTATE NEXTSCOPE) of STATE))
          (SETQ BUILD.CURRENT.SCOPE (fetch (PARSERSTATE CURRENTSCOPE) of STATE))
          (SETQ BUILD.SCOPE.STACK (fetch (PARSERSTATE SCOPESTACK) of STATE))
          (SETQ BUILD.FILECOMS (fetch (PARSERSTATE FILECOMS) of STATE])
```

(**PARSE.BIN**

```
  (CL:LAMBDA (EXPECTCLASS)                                         ; Edited 10-Apr-87 16:00 by Masinter
         (PROG (OLDATOM TOKEN)
               (COND
                  ([AND EXPECTCLASS (OR (AND (LITATOM EXPECTCLASS)
                                             (NOT (EQ EXPECTCLASS PARSE.CLASS)))
                                        (AND (LISTP EXPECTCLASS)
                                             (NOT (FMEMB PARSE.CLASS EXPECTCLASS]
                   (SHOULDNT "PARSE.BIN")))
               (SETQ OLDATOM PARSE.ATOM)
               (SETQ TOKEN (SCAN.TOKEN PARSE.STREAM))
               (SETQ PARSE.CLASS PARSE.CLASS2)
               (SETQ PARSE.ATOM PARSE.ATOM2)
               (SETQ PARSE.CLASS2 (CAR TOKEN))
               (SETQ PARSE.ATOM2 (CADR TOKEN))
               (RETURN OLDATOM))))
```

(**PARSE.VARID**

```
  [LAMBDA NIL                                                      (* kbr%: "25-Nov-85 12:46")
    (BUILD.VARID NIL (PARSE.BIN 'ID])
```

(**PARSE.SMURF**

```
  [LAMBDA (N)                                                      (* kbr%: "25-Nov-85 12:46")
                                                                   (* Indicate where error occurred while reading file *)
    (COND
       ((NULL N)
        (SETQ N 100)))
    (RESETLST
        (PROG (POSITION START FINISH)                              (* Broken file = previous input file *)
              (SETQ POSITION (GETFILEPTR PARSE.STREAM))
              (RESETSAVE NIL (LIST 'SETFILEPTR PARSE.STREAM POSITION))
              (SETQ START (IMAX 0 (IDIFFERENCE (SUB1 POSITION)
                                          N)))
              (SETQ FINISH (IMIN (GETEOFPTR PARSE.STREAM)
                                 (IPLUS (SUB1 POSITION)
                                        N)))
              (COPYBYTES PARSE.STREAM T START (SUB1 POSITION))
              (PRIN1 "[PARSE]" T)
              (COPYBYTES PARSE.STREAM T (SUB1 POSITION)
                     FINISH)
              (TERPRI T)))])
```

(**PARSE.THISIS.MESA**

```
  [LAMBDA NIL                                                      (* kbr%: "25-Nov-85 12:46")
                                                                   (* Assert this is MESA *)
    (COND
       ((NOT (EQ PARSE.LANGUAGE 'MESA))
        (SHOULDNT])
```

(**PARSE.THISIS.CEDAR**

```
  [LAMBDA NIL                                                      (* kbr%: "25-Nov-85 12:46")
                                                                   (* Assert this is CEDAR *)
    (COND
       ((NOT (EQ PARSE.LANGUAGE 'CEDAR))
        (SHOULDNT])
```

(**PARSE.MODULE**

```
  [LAMBDA NIL                                                      (* kbr%: "25-Nov-85 12:46")
    (PROG (IDENTLIST)                                              (* (module directory identlist cedar proghead trusted checked
                                                                   block) (module directory identlist cedar defhead defbody) *)
          (PARSE.DIRECTORY)
          (SETQ IDENTLIST (PARSE.IDENTLIST))
          (BUILD.INIT (CAR IDENTLIST))
          (BUILD.STORE.INTERFACES IDENTLIST)
          (PARSE.SEADIRT)
          (COND
             ((NOT (EQ PARSE.ATOM 'DEFINITIONS))
              (PARSE.PROGHEAD)
              (PARSE.CHECKED)
              (PARSE.BLOCK))
             (T (PARSE.DEFHEAD)
                (PARSE.DEFBODY)))
          (PUTPROP BUILD.PREFIX 'MESA.PARSED T)
          (pushnew PARSE.FILELST BUILD.PREFIX])
```

```
(PARSE.INCLUDEITEM
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 12:46")
    (PROG (ID USING)                                                 (* (includeitem id %: FROM string using)
                                                                     (includeitem id %: TYPE using) (includeitem id using)
                                                                     (includeitem id %: TYPE id using) *)
          (SETQ ID (PARSE.BIN 'ID))
          (BUILD.STORE.INTERFACE ID)
          (PARSE.INCLUDECHECK ID)
          [COND
             [(EQ PARSE.ATOM '%:)
              (PARSE.BIN)
              (COND
                 ((EQ PARSE.ATOM 'FROM)
                  (PARSE.BIN)
                  (PARSE.BIN 'STRING)
                  (SETQ USING (PARSE.USING)))
                 (T (PARSE.BIN 'TYPE)
                    (COND
                       ((EQ PARSE.ATOM 'ID)
                        (PARSE.BIN 'ID)
                        (SETQ USING (PARSE.USING)))
                       ((EQ PARSE.ATOM 'USING)
                        (SETQ USING (PARSE.USING]
             (T (SETQ USING (PARSE.USING]
          (BUILD.STORE.USING ID USING])

(PARSE.INCLUDECHECK
  [LAMBDA (ID)                                                       (* kbr%: "25-Nov-85 12:46")
    (PROG (STATE FILE)
          (COND
             ((GETPROP ID 'MESA.PARSED)                              (* Interface already loaded. *)
              (RETURN)))
          (SELECTQ (ASKUSER NIL NIL (CONCAT "Should I parse " ID ".MESA?"))
              (Y [SETQ FILE (OR (FINDFILE (PACK* ID '.MESA)
                                       NIL PARSE.DIRLST)
                                (MKATOM (PROMPTFORWORD (CONCAT "Enter full filename for " ID ".MESA:"]
                 (COND
                    (FILE (SETQ STATE (PARSE.GET.STATE))
                          (PARSE.FILE FILE PARSE.LANGUAGE PARSE.DIRLST)
                          (PARSE.SET.STATE STATE))))
              (N NIL)
              (SHOULDNT])

(PARSE.SEADIRT
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 12:46")
                                                                     (* BOTH (cedar) *)
                                                                     (* CEDAR (cedar CEDAR) *)
    (COND
       ((EQ PARSE.ATOM 'CEDAR)
        (PARSE.THISIS.CEDAR)
        (PARSE.BIN])

(PARSE.PROGHEAD
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 12:46")
    (PROG NIL                                                        (* (proghead resident safe class arguments locks interface tilde
     public) *)                                                      (* In MESA, tilde must be =. This is handled by PARSE.TILDE.
                                                                     *)
          (PARSE.RESIDENT)
          (PARSE.SAFE)
          (PARSE.CLASS)
          (PARSE.ARGUMENTS)
          (PARSE.LOCKS)
          (PARSE.INTERFACE)
          (PARSE.TILDE)
          (PARSE.PUBLIC])

(PARSE.RESIDENT
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 12:47")
                                                                     (* BOTH (resident) *)
                                                                     (* MESA (resident RESIDENT) *)
    (COND
       ((EQ PARSE.ATOM 'RESIDENT)
        (PARSE.THISIS.MESA)
        (PARSE.BIN])

(PARSE.SAFE
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 12:47")
                                                                     (* BOTH (safe) *)
                                                                     (* CEDAR (safe UNSAFE) (safe SAFE) *)
    (COND
        ((FMEMB PARSE.ATOM '(SAFE UNSAFE))
         (PARSE.THISIS.CEDAR)
         (PARSE.BIN])

(PARSE.DEFHEAD
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG NIL                                                    (* (defhead definitions locks imports shares tilde public) *)
          (PARSE.DEFINITIONS)
          (PARSE.LOCKS)
          (PARSE.IMPORTS)
          (PARSE.SHARES)
          (PARSE.TILDE)
          (PARSE.PUBLIC])

(PARSE.TILDE
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
                                                                 (* BOTH (tilde =) *)
                                                                 (* CEDAR (tilde ~) *)
    (COND
       ((EQ PARSE.ATOM '=)
        (PARSE.BIN))
       ((EQ PARSE.ATOM '~)
        (PARSE.THISIS.CEDAR)
        (PARSE.BIN))
       (T (SHOULDNT])

(PARSE.DEFINITIONS
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
                                                                 (* (definitions DEFINITIONS) *)
    (PARSE.BIN])

(PARSE.DEFBODY
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG NIL                                                    (* (defbody BEGIN open declist END)
                                                                 (defbody BEGIN open declist ; END)
                                                                 (defbody { open declist }) (defbody { open declist ;
                                                                 }) *)
          (PARSE.BIN '(BEGIN {))
          (BUILD.PUSH.SCOPE)
          (BUILD.STORE.OPEN (PARSE.OPEN))
          (PARSE.DECLIST)
          (BUILD.POP.SCOPE)
          (BUILD.GC.SCOPE)
          (COND
             ((EQ PARSE.ATOM ';)
              (PARSE.BIN)))
          (PARSE.BIN '(END }])

(PARSE.LOCKS
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG NIL                                                    (* (locks LOCKS primary lambda)
                                                                 (locks) *)
          (COND
             ((EQ PARSE.ATOM 'LOCKS)
              (PARSE.BIN)
              (PARSE.PRIMARY)
              (PARSE.LAMBDA])

(PARSE.LAMBDA
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (IDENT TYPEEXP)                                        (* (lambda USING ident typeexp)
                                                                 (lambda) *)
          (COND
             ((EQ PARSE.ATOM 'USING)
              (PARSE.BIN)
              (SETQ IDENT (PARSE.IDENT))
              (SETQ TYPEEXP (PARSE.TYPEEXP))
              (BUILD.INITIALIZE.VAR IDENT TYPEEXP NIL BUILD.CURRENT.SCOPE])

(PARSE.MODULEITEM
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (ID1 ID2)                                              (* (moduleitem id) (moduleitem id %: id) *)
          (SETQ ID1 (PARSE.BIN 'ID))
          [COND
             ((EQ PARSE.ATOM '%:)
              (PARSE.BIN)
              (SETQ ID2 (PARSE.BIN 'ID))
              (PUTPROP ID1 'MESA.ABBREVIATES 'ID2]
          (RETURN ID1])
```

```
(PARSE.DECLARATION
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
    (PROG (IDENTLIST TYPEEXP INITIALIZATION DEFAULT OPTSIZE ANSWER)
                                                                  (* (declaration identlist public entry readonly typeexp
                                                                  initialization) (declaration identlist public TYPE tilde public
                                                                  typeexp default) (declaration identlist public TYPE optsize) *)
                                                                  (* In MESA, tilde must be =. This is handled by PARSE.TILDE.
                                                                  *)
          (SETQ IDENTLIST (PARSE.IDENTLIST))
          (BUILD.STORE.IDENTLIST IDENTLIST)
          (PARSE.PUBLIC)
          [COND
             ((NOT (EQ PARSE.ATOM 'TYPE))
              (PARSE.ENTRY)
              (PARSE.READONLY)
              (SETQ TYPEEXP (PARSE.TYPEEXP))
              (SETQ INITIALIZATION (PARSE.INITIALIZATION))
              (SETQ ANSWER (BUILD.INITIALIZATION IDENTLIST TYPEEXP INITIALIZATION)))
             (T (PARSE.BIN 'TYPE)
                (COND
                   ([OR (EQ PARSE.ATOM '=)
                        (AND (EQ PARSE.LANGUAGE 'CEDAR)
                             (EQ PARSE.ATOM '~]
                    (PARSE.TILDE)
                    (PARSE.PUBLIC)
                    (SETQ TYPEEXP (PARSE.TYPEEXP))
                    (SETQ DEFAULT (PARSE.DEFAULT))
                    (BUILD.TYPE IDENTLIST TYPEEXP DEFAULT))
                   (T (SETQ OPTSIZE (PARSE.OPTSIZE))

          (* I think this means MESA/CEDAR is to treat declared id as a type, but no declaration of id is given in this file.
          *)

                      ]
          (BUILD.STORE.IDENTLIST NIL)
          (RETURN ANSWER])

(PARSE.PUBLIC
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
                                                                  (* (public PUBLIC) (public PRIVATE)
                                                                  (public) *)
    (COND
       ((MEMB PARSE.ATOM '(PUBLIC PRIVATE))
        (PARSE.BIN])

(PARSE.ENTRY
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
                                                                  (* (entry ENTRY) (entry INTERNAL)
                                                                  (entry) *)
    (COND
       ((MEMB PARSE.ATOM '(ENTRY INTERNAL))
        (PARSE.BIN])

(PARSE.IDLIST
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
                                                                  (* (idlist' id) (idlist' id %, idlist') *)
    (PROG (IDS ANSWER)
          (push IDS (PARSE.BIN 'ID))
          [while (EQ PARSE.ATOM '%,) do (PARSE.BIN)
                                        (push IDS (PARSE.BIN 'ID]
          (SETQ ANSWER (DREVERSE IDS))
          (RETURN ANSWER])

(PARSE.IDENTLIST
  [LAMBDA (KIND)                                                  (* kbr%: "25-Nov-85 12:47")
                                                                  (* (identlist' id %:) (identlist' id position %:)
                                                                  (identlist' id %, identlist') (identlist' id position %, identlist') *)
    (PROG (IDS TYPEITEMS ANSWER)
      LOOP
          (COND
             ((AND (EQ KIND 'FIELDLIST)
                   (PARSE.TYPEEXP.HERE))

          (* Thought we we're parsing a pairlist, but now we learn we are in a typelist.
          *)

              (SETQ TYPEITEMS (fetch (TYPELIST ITEMS) of (PARSE.TYPELIST)))
              (GO TYPELIST)))
          (push IDS (PARSE.BIN 'ID))
          (COND
             ((EQ PARSE.ATOM '%()
              (PARSE.POSITION)))
```

```
         (COND
            ((EQ PARSE.ATOM '%,)
             (PARSE.BIN)
             (GO LOOP))
            (T (GO EXIT)))
         (GO LOOP)
     EXIT
         (COND
            ((NOT (EQ PARSE.ATOM '%:))
             (GO TYPELIST)))
         (PARSE.BIN '%:)
         (SETQ ANSWER (DREVERSE IDS))
         (RETURN ANSWER)
     TYPELIST
         (SETQ ANSWER (create TYPELIST
                             ITEMS _ (NCONC (DREVERSE IDS)
                                            TYPEITEMS)))
         (RETURN ANSWER])
```

(PARSE.POSITION

```
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (EXP OPTBITS ANSWER)                                   (* (position %( exp optbits %)) *)
         (PARSE.BIN '%()
         (SETQ EXP (PARSE.EXP))
         (SETQ OPTBITS (PARSE.OPTBITS))
         (PARSE.BIN '%))
         (SETQ ANSWER (LIST 'position EXP OPTBITS))
         (RETURN ANSWER])
```

(PARSE.OPTBITS

```
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
                                                                 (* (optbits %: bounds) (optbits) *)
    (COND
       ((EQ PARSE.ATOM '%:)
        (PARSE.BIN '%:)
        (PARSE.BOUNDS])
```

(PARSE.INTERVAL

```
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (KIND BOUNDS ANSWER)                                   (* (interval %[ bounds %]) (interval %[ bounds %))
                                                                 (interval %( bounds %]) (interval %( bounds %)) *)
         (SELECTQ PARSE.ATOM
              (%[ (PARSE.BIN)
                  (SETQ BOUNDS (PARSE.BOUNDS))
                  (SELECTQ PARSE.ATOM
                       (%] (SETQ KIND 'CC))
                       (%) (SETQ KIND 'CO))
                       (SHOULDNT))
                  (PARSE.BIN))
              (%( (PARSE.BIN)
                  (SETQ BOUNDS (PARSE.BOUNDS))
                  (SELECTQ PARSE.ATOM
                       (%] (SETQ KIND 'OC))
                       (%) (SETQ KIND 'OO))
                       (SHOULDNT))
                  (PARSE.BIN))
              (SHOULDNT))
         (SETQ ANSWER (create MINTERVAL
                             KIND _ KIND
                             BOUNDS _ BOUNDS))
         (RETURN ANSWER])
```

(PARSE.TYPEEXP.HERE

```
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    NIL])
```

(PARSE.TYPEEXP

```
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (ANSWER)                                               (* (typeexp id) (typeexp typeid) (typeexp typecons))
         [COND
            [(EQ PARSE.CLASS 'ID)
             (SETQ ANSWER (PARSE.BIN))
             [COND
                ((NOT (FMEMB PARSE.ATOM PARSE.TYPEEXP.FOLLOW))
                 (SETQ ANSWER (PARSE.TYPEID.CONT ANSWER)))
                (T (SETQ ANSWER (BUILD.TYPEID NIL ANSWER]
             (COND
                ((NOT (FMEMB PARSE.ATOM PARSE.TYPEEXP.FOLLOW))
                 (SETQ ANSWER (PARSE.TYPECONS.CONT ANSWER]
            (T (SETQ ANSWER (PARSE.TYPECONS]
         (RETURN ANSWER])
```

```
(PARSE.RANGE
  [LAMBDA NIL                                                  (* kbr%: "25-Nov-85 12:47")
    (PROG (TYPE INTERVAL ANSWER)                               (* (range id) (range id interval) (range typeid interval)
                                                               (range interval) (range typeid) *)
          [COND
             ((FMEMB PARSE.ATOM PARSE.INTERVAL.FIRST)
              (SETQ TYPE 'CARDINAL)
              (SETQ INTERVAL (PARSE.INTERVAL)))
             ((FMEMB PARSE.ATOM2 PARSE.RANGE.FOLLOW)           (* This case occurs if TYPE itself is a range type.
                                                               *)
              [SETQ TYPE (BUILD.TYPEID NIL (PARSE.BIN 'ID]
              (RETURN TYPE))
             ((FMEMB PARSE.ATOM2 PARSE.INTERVAL.FIRST)
              [SETQ TYPE (BUILD.TYPEID NIL (PARSE.BIN 'ID]
              (SETQ INTERVAL (PARSE.INTERVAL)))
             (T (SETQ TYPE (PARSE.TYPEID))
                (COND
                   ((FMEMB PARSE.ATOM PARSE.INTERVAL.FIRST)
                    (SETQ INTERVAL (PARSE.INTERVAL]
          (SETQ ANSWER (create MRANGE
                               TYPE _ TYPE
                               INTERVAL _ INTERVAL))
          (RETURN ANSWER])

(PARSE.TYPEAPPL
  [LAMBDA NIL                                                  (* kbr%: "25-Nov-85 12:47")
    (PROG NIL                                                  (* (typeappl typeappl %. id) (typeappl id length)
                                                               (typeappl typeid length) (typeappl typeappl length) *)
          (BREAK1 NIL T])

(PARSE.TYPEAPPL.CONT
  [LAMBDA (TYPEAPPL)                                           (* kbr%: "25-Nov-85 12:47")
    (PROG (ID LENGTH ANSWER)
          (SETQ ANSWER TYPEAPPL)
          [while (FMEMB PARSE.ATOM '(%. %[)) do (COND
                                                   ((EQ PARSE.ATOM '%.)
                                                    (PARSE.BIN)
                                                    (SETQ ID (PARSE.BIN 'ID))
                                                    (SETQ ANSWER (LIST ANSWER ID)))
                                                   (T (SETQ LENGTH (PARSE.LENGTH))
                                                      (SETQ ANSWER (LIST ANSWER LENGTH]
          (RETURN ANSWER])

(PARSE.TYPEID
  [LAMBDA NIL                                                  (* kbr%: "25-Nov-85 12:47")
    (PARSE.TYPEID.CONT (PARSE.BIN 'ID])

(PARSE.TYPEID.CONT
  [LAMBDA (ID)                                                 (* kbr%: "25-Nov-85 12:47")
    (PROG (INTERFACE ANSWER)                                   (* (typeid' id %. id) (typeid' typeid' %.
                                                               id) (typeid id id) (typeid id typeid)
                                                               (typeid typeid') *)
                                                               (* Should be ID+{.ID}* *)
          (while (EQ PARSE.CLASS 'ID) do (BREAK1 NIL T)
                                         (SETQ ID (PARSE.BIN)))
          [COND
             ((EQ PARSE.ATOM '%.)
              (SETQ INTERFACE ID)
              (PARSE.BIN)
              (SETQ ID (PARSE.BIN 'ID]
          (SETQ ANSWER (BUILD.TYPEID INTERFACE ID))
          (RETURN ANSWER])

(PARSE.TYPECONS
  [LAMBDA NIL                                                  (* kbr%: "25-Nov-85 12:47")
    (COND
       ((EQ PARSE.CLASS 'ID)
        (PARSE.TYPECONS1))
       (T (PARSE.TYPECONS2])

(PARSE.TYPECONS1
  [LAMBDA NIL                                                  (* kbr%: "25-Nov-85 12:47")
                                                               (* TYPECONS begining with ID token.
                                                               *)
    (PROG (TYPEID ANSWER)                                      (* BOTH (typecons id interval) (typecons typeid interval)
                                                               (typecons id RELATIVE typeexp)
                                                               (typecons typeid RELATIVE typeexp)
                                                               (typecons typeappl) *)
                                                               (* CEDAR (typecons id PAINTED typeexp)
                                                               (typecons typeid PAINTED typeexp) *)
```

```
                                                                          (* Get id or typeid. *)
          (SETQ TYPEID (PARSE.BIN 'ID))
          [COND
             ((NOT (FMEMB PARSE.ATOM PARSE.TYPEEXP.FOLLOW))
              (SETQ TYPEID (PARSE.TYPEID.CONT TYPEID)))
             (T (SETQ TYPEID (BUILD.TYPEID NIL TYPEID]                    (* Finish typecons. *)
          (SETQ ANSWER (PARSE.TYPECONS.CONT TYPEID))
          (RETURN ANSWER])
```

(**PARSE.TYPECONS.CONT**

```
  [LAMBDA (TYPEID)                                                        (* kbr%: "25-Nov-85 12:47")
                                                                          ; TYPEID is an id or typeid.  Finish typecons.
    (PROG (INTERVAL TYPEEXP EXP1 EXP2 KIND ANSWER)

     ;; BOTH (typecons id interval) (typecons typeid interval) (typecons id RELATIVE typeexp) (typecons typeid RELATIVE typeexp) (typecons
     ;; typeappl)

     ;; CEDAR (typecons id PAINTED typeexp) (typecons typeid PAINTED typeexp) *

          (COND
             ((EQ PARSE.ATOM 'RELATIVE)
              (SETQ ANSWER (PARSE.TYPECONS.RELATIVE)))
             ((EQ PARSE.ATOM 'PAINTED)
              (SETQ ANSWER (PARSE.TYPECONS.PAINTED)))
             ((EQ PARSE.ATOM '%()
              (PARSE.TYPECONS.RANGE TYPEID))
             [(EQ PARSE.ATOM '%[)                                         ; This can be the start of a length or of an interval.  Can't tell with
                                                                          ; bounded look ahead.
              (PARSE.BIN '%[)
              (SETQ EXP1 (PARSE.EXP))
              (COND
                 ((EQ PARSE.ATOM '|..|)                                   ; Interval.
                  (PARSE.BIN '|..|)
                  (SETQ EXP2 (PARSE.EXP))
                  [COND
                     ((EQ PARSE.ATOM '%))
                      (PARSE.BIN '%))
                      (SETQ KIND 'CO))
                     (T (PARSE.BIN '%])
                        (SETQ KIND 'CC]
                  (SETQ INTERVAL (create MINTERVAL
                                        KIND _ KIND
                                        BOUNDS _ (LIST EXP1 EXP2)))
                  (SETQ ANSWER (create MRANGE
                                      TYPE _ TYPEID
                                      INTERVAL _ INTERVAL)))
                 (T                                                       ; Length.  *
                    (PARSE.BIN '%])
                    (SETQ ANSWER (LIST TYPEID EXP1))
                    (SETQ ANSWER (PARSE.TYPEAPPL.CONT ANSWER]
             (T (SHOULDNT)))
          (RETURN ANSWER])
```

(**PARSE.TYPECONS.RANGE**

```
  [LAMBDA (TYPEID)                                                        (* kbr%: "25-Nov-85 12:47")
    (PROG (INTERVAL ANSWER)
          (SETQ INTERVAL (PARSE.INTERVAL))
          (SETQ ANSWER (create MRANGE
                              TYPE _ TYPEID
                              INTERVAL _ INTERVAL))
          (RETURN ANSWER])
```

(**PARSE.TYPECONS.RELATIVE**

```
  [LAMBDA (TYPEID)                                                        (* kbr%: "25-Nov-85 12:47")
    (PROG (TYPE ANSWER)
          (PARSE.BIN 'RELATIVE)
          (SETQ TYPE (PARSE.TYPEEXP))
          (SETQ ANSWER (create MRELATIVE
                              TYPEID _ TYPEID
                              TYPE _ TYPE))
          (RETURN ANSWER])
```

(**PARSE.TYPECONS.PAINTED**

```
  [LAMBDA (TYPEID)                                                        (* kbr%: "25-Nov-85 12:47")
    (PROG (TYPE ANSWER)
          (PARSE.THISIS.CEDAR)
          (PARSE.BIN 'RELATIVE)
          (SETQ TYPE (PARSE.TYPEEXP))
          (SETQ ANSWER (create MPAINTED
                              TYPEID _ TYPEID
                              TYPE _ TYPE))
          (RETURN ANSWER])
```

(**PARSE.TYPECONS2**

```
  [LAMBDA NIL                                                                  (* kbr%: "25-Nov-85 12:47")
                                                                               (* TYPECONS begining with reserved word.
                                                                               *)
    (PROG (ANSWER)                                                             (* BOTH (typecons interval) (typecons dependent { elementlist })
                                                                               (typecons dependent monitored RECORD reclist)
                                                                               (typecons ordered base pointertype)
                                                                               (typecons VAR typeexp) (typecons packed ARRAY indextype
                                                                               OF typeexp) (typecons DESCRIPTOR FOR readonly typeexp)
                                                                               (typecons safe transfermode arguments)
                                                                               (typecons heap ZONE) (typecons LONG typeexp)
                                                                               (typecons FRAME %[ id %]) *)
                                                                               (* CEDAR (typecons REF readonly typeexp)
                                                                               (typecons REF readonly ANY) (typecons REF)
                                                                               (typecons LIST OF readonly typeexp) *)
          [SETQ ANSWER (COND
                          ((FMEMB PARSE.ATOM PARSE.INTERVAL.FIRST)
                           (PARSE.TYPECONS.INTERVAL))
                          (T (SELECTQ PARSE.ATOM
                                 ((MACHINE MONITORED RECORD {)
                                      (PARSE.TYPECONS.DEPENDENT))
                                 ((ORDERED BASE POINTER)
                                      (PARSE.TYPECONS.ORDERED))
                                 (VAR (PARSE.TYPECONS.VAR))
                                 ((PACKED ARRAY)
                                      (PARSE.TYPECONS.PACKED))
                                 (DESCRIPTOR (PARSE.TYPECONS.DESCRIPTOR))
                                 ((SAFE ERROR PORT PROC PROCEDURE PROCESS PROGRAM SIGNAL)
                                      (PARSE.TYPECONS.SAFE))
                                 (UNCOUNTED (PARSE.TYPECONS.HEAP))
                                 (LONG (PARSE.TYPECONS.LONG))
                                 (FRAME (PARSE.TYPECONS.FRAME))
                                 (REF (PARSE.TYPECONS.REF))
                                 (LIST (PARSE.TYPECONS.LIST))
                                 (SHOULDNT]
          (RETURN ANSWER])
```

(PARSE.TYPECONS.INTERVAL

```
  [LAMBDA NIL                                                                  (* kbr%: "25-Nov-85 12:47")
    (PROG (ANSWER)                                                             (* (typecons interval) *)
          (SETQ ANSWER (create MRANGE
                              TYPE _ 'CARDINAL
                              INTERVAL _ (PARSE.INTERVAL)))
          (RETURN ANSWER])
```

(PARSE.TYPECONS.DEPENDENT

```
  [LAMBDA NIL                                                                  (* kbr%: "25-Nov-85 12:47")
    (PROG (ELEMENTLIST RECLIST ANSWER)                                         (* (typecons dependent { elementlist })
                                                                               (typecons dependent monitored RECORD reclist) *)
          (PARSE.DEPENDENT)
          [SETQ ANSWER (COND
                          ((EQ PARSE.ATOM '{)
                           (PARSE.TYPECONS.ENUMERATED))
                          (T (PARSE.TYPECONS.RECORD]
          (RETURN ANSWER])
```

(PARSE.TYPECONS.ENUMERATED

```
  [LAMBDA NIL                                                                  (* kbr%: "25-Nov-85 12:47")
    (PROG (ITEMS ANSWER)
          (PARSE.BIN)
          (SETQ ITEMS (PARSE.ELEMENTLIST))
          (PARSE.BIN '})
          (SETQ ANSWER (create MENUMERATED
                              ITEMS _ ITEMS))
          (RETURN ANSWER])
```

(PARSE.TYPECONS.RECORD

```
  [LAMBDA NIL                                                                  (* kbr%: "25-Nov-85 12:47")
    (PROG (FIELDLIST ANSWER)
          (PARSE.MONITORED)
          (PARSE.BIN 'RECORD)
          (SETQ FIELDLIST (PARSE.RECLIST))
          (SETQ ANSWER (create MRECORD
                              FIELDLIST _ FIELDLIST))
          (RETURN ANSWER])
```

(PARSE.TYPECONS.ORDERED

```
  [LAMBDA NIL                                                                  (* kbr%: "25-Nov-85 12:47")
    (PROG (ANSWER)                                                             (* (typecons ordered base pointertype) *)
          (PARSE.ORDERED)
          (PARSE.BASE)
          (SETQ ANSWER (PARSE.POINTERTYPE))
          (RETURN ANSWER])
```

```
(PARSE.TYPECONS.VAR
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
    (PROG (TYPE ANSWER)                                           (* (typecons VAR typeexp) *)
          (PARSE.BIN 'VAR)
          (SETQ TYPE (PARSE.TYPEEXP))
          (SETQ ANSWER (create MVAR
                               TYPE _ TYPE))
          (RETURN ANSWER])

(PARSE.TYPECONS.PACKED
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
    (PROG (PACKED INDEXTYPE TYPE ANSWER)                          (* (typecons packed ARRAY indextype OF typeexp) *)
          (SETQ PACKED (PARSE.PACKED))
          (PARSE.BIN 'ARRAY)
          (SETQ INDEXTYPE (PARSE.INDEXTYPE))
          (PARSE.BIN 'OF)
          (SETQ TYPE (PARSE.TYPEEXP))
          (SETQ ANSWER (create MARRAY
                               INDEXTYPE _ INDEXTYPE
                               TYPE _ TYPE))
          (RETURN ANSWER])

(PARSE.TYPECONS.DESCRIPTOR
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
    (PROG (TYPE ANSWER)                                           (* (typecons DESCRIPTOR FOR readonly typeexp) *)
          (PARSE.BIN 'DESCRIPTOR)
          (PARSE.BIN 'FOR)
          (PARSE.READONLY)
          (SETQ TYPE (PARSE.TYPEEXP))
          (SETQ ANSWER (create MDESCRIPTOR
                               TYPE _ TYPE))
          (RETURN ANSWER])

(PARSE.TYPECONS.SAFE
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
    (PROG (TRANSFERMODE)                                          (* (typecons safe transfermode arguments) *)
          (PARSE.SAFE)
          (SETQ TRANSFERMODE (PARSE.TRANSFERMODE))
          (PARSE.ARGUMENTS)
          (RETURN TRANSFERMODE])

(PARSE.TYPECONS.HEAP
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
    (PROG NIL                                                     (* (typecons heap ZONE) *)
          (PARSE.HEAP)
          (PARSE.BIN 'ZONE)
          (RETURN 'ZONE])

(PARSE.TYPECONS.LONG
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
    (PROG (ANSWER)                                                (* (typecons LONG typeexp) *)
          (PARSE.BIN 'LONG)
          (SETQ ANSWER (PARSE.TYPEEXP))
          (RETURN ANSWER])

(PARSE.TYPECONS.FRAME
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
    (PROG (ID ANSWER)                                             (* (typecons FRAME %[ id %]) *)
          (PARSE.BIN 'FRAME)
          (PARSE.BIN '%[)
          [SETQ ID (BUILD.ID NIL (PARSE.BIN 'ID]
          (PARSE.BIN '%])
          (SETQ ANSWER (create MFRAME
                               ID _ ID))
          (RETURN ANSWER])

(PARSE.TYPECONS.REF
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
    (PROG (TYPE ANSWER)                                           (* CEDAR (typecons REF readonly typeexp)
                                                                  (typecons REF readonly ANY) (typecons REF) *)

          (PARSE.THISIS.CEDAR)
          (PARSE.BIN 'REF)
          [COND
             ((FMEMB PARSE.ATOM PARSE.TYPEEXP.FOLLOW)
              (SETQ TYPE 'ANY))
             (T (PARSE.READONLY)
                (COND
                   ((EQ PARSE.ATOM 'ANY)
                     (PARSE.BIN)
                     (SETQ TYPE 'ANY))
                    (T (SETQ TYPE (PARSE.TYPEEXP]
          (SETQ ANSWER (create MREF
                               TYPE _ TYPE))
          (RETURN ANSWER])
```

(PARSE.TYPECONS.LIST

```
  [LAMBDA NIL                                                 (* kbr%: "25-Nov-85 12:47")
    (PROG (TYPE ANSWER)                                       (* CEDAR (typecons LIST OF readonly typeexp) *)
          (PARSE.THISIS.CEDAR)
          (PARSE.BIN 'LIST)
          (PARSE.BIN 'OF)
          (PARSE.READONLY)
          (SETQ TYPE (PARSE.TYPEEXP))
          (SETQ ANSWER (create MLIST
                               TYPE _ TYPE))
          (RETURN ANSWER])
```

(PARSE.IDENT

```
  [LAMBDA NIL                                                 (* kbr%: "25-Nov-85 12:47")
    (PROG (ID)                                                (* (ident id position %:) (ident id %:) *)
          (SETQ ID (PARSE.BIN 'ID))
          [COND
             ((EQ PARSE.ATOM '%:)
              (PARSE.BIN))
             (T (PARSE.POSITION)
                (PARSE.BIN '%:]
          (RETURN ID])
```

(PARSE.ELEMENT

```
  [LAMBDA NIL                                                 (* kbr%: "25-Nov-85 12:47")
    (PROG (ID EXP ANSWER)                                     (* (element id %( exp %)) (element %( exp %))
                                                              (element id) *)
          [COND
             ((NOT (EQ PARSE.ATOM '%())
              (SETQ ID (PARSE.BIN 'ID]
          (COND
             ((EQ PARSE.ATOM '%()
              (PARSE.BIN)
              (SETQ EXP (PARSE.EXP))
              (SETQ ANSWER ID)
              (PARSE.BIN '%)))
             (T (SETQ ANSWER ID)))
          (RETURN ANSWER])
```

(PARSE.MONITORED

```
  [LAMBDA NIL                                                 (* kbr%: "25-Nov-85 12:47")
                                                              (* (monitored MONITORED) (monitored) *)
    (COND
       ((EQ PARSE.ATOM 'MONITORED)
        (PARSE.BIN])
```

(PARSE.DEPENDENT

```
  [LAMBDA NIL                                                 (* kbr%: "25-Nov-85 12:47")
                                                              (* (dependent MACHINE DEPENDENT)
                                                              (dependent) *)
    (COND
       ((EQ PARSE.ATOM 'MACHINE)
        (PARSE.BIN)
        (PARSE.BIN 'DEPENDENT)
        'MACHINE.DEPENDENT])
```

(PARSE.RECLIST

```
  [LAMBDA NIL                                                 (* kbr%: "25-Nov-85 12:47")
    (PROG (PAIRLIST TYPELIST VARIANTPAIR VARIANTPART DEFAULT ANSWER)
                                                              (* (reclist %[ %]) (reclist NULL) (reclist %[ pairlist %])
                                                              (reclist %[ typelist %]) (reclist %[ pairlist %, variantpair %])
                                                              (reclist %[ variantpart default %])
                                                              (reclist %[ variantpair %]) *)
          (COND
             ((EQ PARSE.ATOM 'NULL)
              (PARSE.BIN)
              (RETURN NIL)))
          (PARSE.BIN '%[)
          (COND
             ((EQ PARSE.ATOM '%])
              (PARSE.BIN)
              (RETURN NIL)))
          (COND
             [(FMEMB PARSE.ATOM PARSE.VARIANTPART.FIRST)
```

```
              (SETQ VARIANTPART (PARSE.VARIANTPART))
              (SETQ DEFAULT (PARSE.DEFAULT))
              (SETQ ANSWER (LIST (create PAIRITEM
                                     TYPEEXP _ VARIANTPART
                                     DEFAULT _ DEFAULT]
             ([AND (EQ PARSE.CLASS 'ID)
                   (NOT (FMEMB PARSE.ATOM PARSE.PREDEFINED.TYPES))
                   (FMEMB PARSE.ATOM2 '(%( %, %:]
              (SETQ PAIRLIST (PARSE.PAIRLIST 'RECLIST))
              [for PAIRITEM in (fetch (PAIRLIST ITEMS) of PAIRLIST) do (replace (PAIRITEM ID) of PAIRITEM
                                                                           with (BUILD.FIELDID NIL
                                                                                       (fetch (PAIRITEM ID)
                                                                                          of PAIRITEM]
              (SETQ ANSWER PAIRLIST))
             (T (SETQ TYPELIST (PARSE.TYPELIST))
                (SETQ ANSWER TYPELIST)))
          (PARSE.BIN '%])
          (RETURN ANSWER])
```

(PARSE.VARIANTPAIR

```
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (IDENTLIST PUBLIC VARIANTPART DEFAULT ANSWER)          (* (variantpair identlist public variantpart default) *)
          (SETQ IDENTLIST (PARSE.IDENTLIST))
          (PARSE.PUBLIC)
          (SETQ VARIANTPART (PARSE.VARIANTPART))
          (SETQ DEFAULT (PARSE.DEFAULT))
          (SETQ ANSWER (for ID in IDENTLIST collect (create PAIRITEM
                                                           ID _ ID
                                                           TYPEEXP _ VARIANTPART
                                                           DEFAULT _ DEFAULT)))
          (RETURN ANSWER])
```

(PARSE.PAIRITEM

```
  [LAMBDA (KIND)                                                 (* kbr%: "25-Nov-85 12:47")
    (PROG (IDENTLIST VARIANTPART TYPEEXP DEFAULT ANSWER)         (* (pairitem identlist public typeexp default)
                                                                 (variantpair identlist public variantpart default) *)
          (SETQ IDENTLIST (PARSE.IDENTLIST KIND))
          (COND
             ((type? TYPELIST IDENTLIST)                         (* Thought we we're parsing a pairlist but found a typelist.
                                                                 *)
              (RETURN IDENTLIST)))
          (PARSE.PUBLIC)
          [COND
             ([AND (FMEMB PARSE.ATOM PARSE.VARIANTPART.FIRST)
                   (OR (NOT (EQ PARSE.ATOM 'PACKED))
                       (NOT (EQ PARSE.ATOM2 'ARRAY]              (* Variantpair. *)
              (COND
                 ((NOT (EQ KIND 'RECLIST))
                  (SHOULDNT)))
              (SETQ TYPEEXP (PARSE.VARIANTPART)))
             (T                                                  (* Typeexp. *)
                (SETQ TYPEEXP (PARSE.TYPEEXP]
          (SETQ DEFAULT (PARSE.DEFAULT))
          (SETQ ANSWER (for ID in IDENTLIST collect (create PAIRITEM
                                                           ID _ ID
                                                           TYPEEXP _ TYPEEXP
                                                           DEFAULT _ DEFAULT)))
          (RETURN ANSWER])
```

(PARSE.DEFAULTOPT

```
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (EXP TRASH ANSWER)                                     (* (defaultopt TRASH) (defaultopt NULL)
                                                                 (defaultopt exp %| TRASH) (defaultopt exp %| NULL)
                                                                 (defaultopt) (defaultopt exp) *)
          [COND
             ((FMEMB PARSE.ATOM '(TRASH NULL))
              (PARSE.BIN)
              (SETQ TRASH T))
             ((NOT (FMEMB PARSE.ATOM PARSE.DEFAULTOPT.FOLLOW))
              (SETQ EXP (PARSE.EXP))
              (COND
                 ((EQ PARSE.ATOM '%|)
                  (PARSE.BIN '%|)
                  (COND
                     ((FMEMB PARSE.ATOM '(TRASH NULL))
                      (PARSE.BIN)
                      (SETQ TRASH T]
          (SETQ ANSWER (create DEFAULT
                               EXP _ EXP
                               TRASH _ TRASH))
          (RETURN ANSWER])
```

(PARSE.VARIANTPART

```
  [LAMBDA NIL                                        (* kbr%: "25-Nov-85 12:47")
    (PROG (VCASEHEAD VARIANTLIST TYPEEXP ANSWER)     (* (variantpart SELECT vcasehead FROM variantlist ENDCASE)
                                                     (variantpart SELECT vcasehead FROM variantlist %,
                                                     ENDCASE) (variantpart packed SEQUENCE vcasehead OF
                                                     typeexp) *)
          [COND
             ((EQ PARSE.ATOM 'SELECT)
              (PARSE.BIN)
              (SETQ VCASEHEAD (PARSE.VCASEHEAD))
              (PARSE.BIN 'FROM)
              (SETQ VARIANTLIST (PARSE.VARIANTLIST))
              (COND
                 ((EQ PARSE.ATOM '%,)
                  (PARSE.BIN)))
              (PARSE.BIN 'ENDCASE)
              (SETQ ANSWER (LIST 'SELECT VCASEHEAD VARIANTLIST)))
             (T (SETQ PACKED (PARSE.PACKED))
                (PARSE.BIN 'SEQUENCE)
                (SETQ VCASEHEAD (PARSE.VCASEHEAD))
                (PARSE.BIN 'OF)
                (SETQ TYPEEXP (PARSE.TYPEEXP))
                (SETQ ANSWER (LIST 'SEQUENCE VCASEHEAD TYPEEXP]
          (RETURN ANSWER])

(PARSE.VCASEHEAD
  [LAMBDA NIL                                        (* kbr%: "25-Nov-85 12:47")
    (PROG (IDENT PUBLIC TAGTYPE ANSWER)              (* (vcasehead ident public tagtype)
                                                     (vcasehead COMPUTED tagtype)
                                                     (vcasehead OVERLAID tagtype) *)
          [COND
             ([NOT (FMEMB PARSE.ATOM '(COMPUTED OVERLAID]
              (SETQ IDENT (PARSE.IDENT))
              (SETQ PUBLIC (PARSE.PUBLIC))
              (SETQ TAGTYPE (PARSE.TAGTYPE))
              (SETQ ANSWER (LIST 'vcasehead IDENT PUBLIC TAGTYPE)))
             (T (SETQ ANSWER (LIST 'vcasehead (PARSE.BIN)
                                   (PARSE.TAGTYPE]
          (RETURN ANSWER])

(PARSE.TAGTYPE
  [LAMBDA NIL                                        (* kbr%: "25-Nov-85 12:47")
                                                     (* (tagtype *) (tagtype typeexp) *)
    (COND
       ((EQ PARSE.ATOM '*)
        (PARSE.BIN))
       (T (PARSE.TYPEEXP])

(PARSE.VARIANTITEM
  [LAMBDA NIL                                        (* kbr%: "25-Nov-85 12:47")
    (PROG (IDLIST RECLIST ANSWER)                    (* (variantitem idlist => reclist) *)
          (SETQ IDLIST (PARSE.IDLIST))
          (PARSE.BIN '=>)
          (SETQ RECLIST (PARSE.RECLIST))
          (SETQ ANSWER (LIST 'variantitem IDLIST RECLIST))
          (RETURN ANSWER])

(PARSE.TYPELIST
  [LAMBDA NIL                                        (* kbr%: "25-Nov-85 12:47")
    (PROG (TYPEITEMS ANSWER)                         (* (typelist typecons default) (typelist typeid default)
                                                     (typelist id) (typelist id _ defaultopt)
                                                     (typelist typecons default %, typelist)
                                                     (typelist typeid default %, typelist)
                                                     (typelist id %, typelist) (typelist id _ defaultopt %, typelist) *)
          (push TYPEITEMS (PARSE.TYPEITEM))
          (while (EQ PARSE.ATOM '%,) do (PARSE.BIN)
                                        (push TYPEITEMS (PARSE.TYPEITEM)))
          (SETQ ANSWER (create TYPELIST
                              ITEMS _ (DREVERSE TYPEITEMS)))
          (RETURN ANSWER])

(PARSE.TYPEITEM
  [LAMBDA NIL                                        (* kbr%: "25-Nov-85 12:47")
    (PROG (TYPEEXP DEFAULT ANSWER)
          (SETQ TYPEEXP (PARSE.TYPEEXP))
          [COND
             ((NOT (LITATOM TYPEEXP))
              (SETQ DEFAULT (PARSE.DEFAULT)))
             ((EQ PARSE.ATOM '_)
              (SETQ DEFAULT (PARSE.DEFAULTOPT]
          (SETQ ANSWER (create TYPEITEM
                              TYPEEXP _ TYPEEXP
                              DEFAULT _ DEFAULT))
          (RETURN ANSWER])

(PARSE.POINTERTYPE
  [LAMBDA NIL                                          (* kbr%: "25-Nov-85 12:47")
    (PROG (TYPE ANSWER)                                (* (pointertype pointerprefix) (pointertype pointerprefix TO
                                                       readonly typeexp) *)
          (PARSE.POINTERPREFIX)
          [COND
             ((EQ PARSE.ATOM 'TO)
              (PARSE.BIN)
              (PARSE.READONLY)
              (SETQ TYPE (PARSE.TYPEEXP)))
             (T (SETQ TYPE 'UNSPECIFIED]
          (SETQ ANSWER (create MPOINTER
                                TYPE _ TYPE))
          (RETURN ANSWER])

(PARSE.TRANSFERMODE
  [LAMBDA NIL                                          (* kbr%: "25-Nov-85 12:47")
                                                       (* (transfermode PROCEDURE) (transfermode PROC)
                                                       (transfermode PORT) (transfermode SIGNAL)
                                                       (transfermode ERROR) (transfermode PROCESS)
                                                       (transfermode PROGRAM) *)
    (PROG NIL
          (PARSE.BIN)
          (RETURN 'PROC])

(PARSE.INITIALIZATION
  [LAMBDA NIL                                          (* kbr%: "25-Nov-85 12:47")
    (PROG (ANSWER)                                     (* (initialization) (initialization _ initvalue)
                                                       (initialization tilde initvalue) *)
                                                       (* In MESA tilde must be =. *)
          [COND
             ([OR (FMEMB PARSE.ATOM '(_ =))
                  (AND (EQ PARSE.LANGUAGE 'CEDAR)
                       (EQ PARSE.ATOM '~]
              (PARSE.BIN)
              (SETQ ANSWER (PARSE.INITVALUE]
          (RETURN ANSWER])

(PARSE.INITVALUE
  [LAMBDA NIL                                          (* kbr%: "25-Nov-85 12:47")
    (PROG (INLINE BLOCK CODELIST EXP ANSWER)           (* (initvalue procaccess trusted checked inline block)
                                                       (initvalue CODE) (initvalue procaccess trusted checked
                                                       MACHINE CODE BEGIN codelist END)
                                                       (initvalue procaccess trusted checked MACHINE CODE {
                                                       codelist }) (initvalue TRASH) (initvalue NULL)
                                                       (initvalue exp) *)
          [COND
             ((FMEMB PARSE.ATOM '(CODE TRASH NULL))
              (PARSE.BIN)
              (SETQ ANSWER 'TRASH))
             (T (PARSE.CHECKED)
                (COND
                   ((FMEMB PARSE.ATOM '(INLINE BEGIN {))
                    (SETQ INLINE (PARSE.INLINE))
                    (SETQ BLOCK (PARSE.BLOCK))
                    (SETQ ANSWER BLOCK))
                   ((EQ PARSE.ATOM 'MACHINE)
                    (PARSE.BIN)
                    (PARSE.BIN 'CODE)
                    (PARSE.BIN '(BEGIN {))
                    (SETQ CODELIST (PARSE.CODELIST))
                    (PARSE.BIN '(END }))
                    (SETQ ANSWER CODELIST))
                   (T (SETQ EXP (PARSE.EXP))
                      (SETQ ANSWER EXP]
          (RETURN ANSWER])

(PARSE.CHECKED
  [LAMBDA NIL                                          (* kbr%: "25-Nov-85 12:47")
                                                       (* BOTH (checked) *)
                                                       (* CEDAR (checked CHECKED) (checked TRUSTED)
                                                       (checked UNCHECKED) *)
    (COND
       ((FMEMB PARSE.ATOM '(CHECKED TRUSTED UNCHECKED))
        (PARSE.THISIS.CEDAR)
        (PARSE.BIN])

(PARSE.CODELIST
  [LAMBDA NIL                                          (* kbr%: "25-Nov-85 12:47")
```

```
    (PROG NIL                                                  (* (codelist orderlist) (codelist codelist ;
                                                               orderlist) *)
          (BREAK1 NIL T])
```

(**PARSE.STATEMENT**

```
  [LAMBDA NIL                                                  (* kbr%: "25-Nov-85 12:47")
    (COND
       ((FMEMB PARSE.CLASS '(ID %())
        (PARSE.STATEMENT1))
       (T (PARSE.STATEMENT2])
```

(**PARSE.STATEMENT1**

```
  [LAMBDA NIL                                                  (* kbr%: "25-Nov-85 12:47")
    (PROG (LHS EXP OPTCATCH ANSWER)                            (* (statement lhs) (statement lhs _ exp)
                                                               (statement free %[ exp optcatch %])
                                                               (statement lhs _ STATE) *)
          (SETQ LHS (PARSE.LHS))
          (COND
             ((AND (EQ PARSE.ATOM '%.)
                   (EQ PARSE.ATOM2 'FREE))
              (PARSE.BIN)
              (PARSE.BIN)
              (PARSE.BIN '%[)
              (SETQ EXP (PARSE.EXP))
              (SETQ OPTCATCH (PARSE.OPTCATCH))
              (PARSE.BIN '%])
              (SETQ ANSWER (LIST LHS EXP OPTCATCH)))
             ((AND (EQ PARSE.ATOM '_)
                   (EQ PARSE.ATOM2 'STATE))
              (PARSE.BIN)
              (PARSE.BIN)
              (SETQ ANSWER LHS))
             ((EQ PARSE.ATOM '_)
              (PARSE.BIN)
              (SETQ EXP (PARSE.EXP))
              (SETQ ANSWER (BUILD.SETQ LHS EXP)))
             (T (SETQ ANSWER LHS)))
          (RETURN ANSWER])
```

(**PARSE.STATEMENT2**

```
  [LAMBDA NIL                                                  (* kbr%: "25-Nov-85 12:47")
    (PROG (ANSWER)

          (* (statement %[ explist %] _ exp) (statement trusted checked block)
          (statement IF exp THEN statement elsepart) (statement casehead casestmtlist ENDCASE otherpart)
          (statement forclause dotest DO scope doexit ENDLOOP) (statement EXIT)
          (statement LOOP) (statement GOTO id) (statement GO TO id)
          (statement RETURN optargs) (statement transfer lhs) (statement WAIT lhs)
          (statement ERROR) (statement STOP) (statement NULL) (statement RESUME optargs)
          (statement REJECT) (statement CONTINUE) (statement RETRY) *)

          [SETQ ANSWER (COND
                          ((FMEMB PARSE.ATOM PARSE.CASEHEAD.FIRST)
                           (PARSE.STATEMENT.CASEHEAD))
                          ((OR (FMEMB PARSE.ATOM PARSE.FORCLAUSE.FIRST)
                               (FMEMB PARSE.ATOM PARSE.DOTEST.FIRST)
                               (EQ PARSE.ATOM 'DO))
                           (PARSE.STATEMENT.FORCLAUSE))
                          ([AND (EQ PARSE.ATOM 'RETURN)
                                (NOT (EQ PARSE.ATOM2 'WITH]    (* Don't confuse statement RETURN with the transfer RETURN
                           WITH. *)
                           (PARSE.STATEMENT.RETURN))
                          ((FMEMB PARSE.ATOM PARSE.TRANSFER.FIRST)
                           (PARSE.STATEMENT.TRANSFER))
                          (T (SELECTQ PARSE.ATOM
                                 (%[ (PARSE.STATEMENT.LBRACKET))
                                 (({ BEGIN CHECKED TRUSTED UNCHECKED)
                                      (PARSE.CHECKED)
                                      (PARSE.BLOCK))
                                 (IF (PARSE.STATEMENT.IF))
                                 (EXIT (PARSE.BIN)
                                       '(RETURN))
                                 (LOOP (PARSE.BIN)
                                       '(GO LOOP))
                                 (GOTO (PARSE.BIN)
                                       (LIST 'GO (PARSE.BIN 'ID)))
                                 (GO (PARSE.BIN)
                                     (PARSE.BIN 'TO)
                                     (LIST 'GO (PARSE.BIN 'ID)))
                                 (WAIT (PARSE.BIN)
                                       (PARSE.LHS))
                                 (ERROR (PARSE.BIN)
                                        '(SHOULDNT))
                                 (STOP (PARSE.BIN)
                                       '(GO STOP))
                                 (NULL (PARSE.BIN)
                                       NIL)
                                 (RESUME (PARSE.BIN)
                                         (PARSE.OPTARGS))
                                 (REJECT (PARSE.BIN)
                                         '(SHOULDNT))
                                 (CONTINUE (PARSE.BIN)
                                           '(GO CONTINUE))
                                 (RETRY (PARSE.BIN)
                                        '(GO RETRY))
                                 (SHOULDNT]
         (RETURN ANSWER])
```

(PARSE.STATEMENT.CASEHEAD

```
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (CASEHEAD CASESTMTLIST OTHERPART ANSWER)               (* (statement casehead casestmtlist ENDCASE otherpart) *)
          (BUILD.PUSH.SCOPE)
          (SETQ CASEHEAD (PARSE.CASEHEAD))
          (SETQ CASESTMTLIST (PARSE.CASESTMTLIST CASEHEAD))
          (PARSE.BIN 'ENDCASE)
          (SETQ OTHERPART (PARSE.OTHERPART))
          (SETQ ANSWER (BUILD.SELECTQ CASEHEAD CASESTMTLIST OTHERPART))
          (COND
             ((fetch (CASEHEAD ID) of CASEHEAD)
              (BUILD.INITIALIZE.VAR (fetch (CASEHEAD ID) of CASEHEAD)
                     NIL
                     (fetch (CASEHEAD EXP) of CASEHEAD)
                     BUILD.CURRENT.SCOPE)))
          (SETQ ANSWER (BUILD.PROG (LIST ANSWER)))
          (BUILD.POP.SCOPE)
          (RETURN ANSWER])
```

(PARSE.STATEMENT.FORCLAUSE

```
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (FORCLAUSE DOTEST SCOPE DOEXIT EXITLIST STATEMENT ANSWER)
                                                                 (* (statement forclause dotest DO scope doexit ENDLOOP) *)
          (BUILD.STORE.IDENTLIST '(DO))
          (BUILD.PUSH.SCOPE)
          (SETQ FORCLAUSE (PARSE.FORCLAUSE))
          (SETQ DOTEST (PARSE.DOTEST))
          (PARSE.BIN 'DO)
          (SETQ SCOPE (PARSE.SCOPE))
          (SETQ DOEXIT (PARSE.DOEXIT))
          (SETQ EXITLIST (CAR DOEXIT))
          (SETQ STATEMENT (CADR DOEXIT))
          (PARSE.BIN 'ENDLOOP)
          (BUILD.POP.SCOPE)
          [SETQ ANSWER `(,@FORCLAUSE ,@DOTEST do ,@(BUILD.TAIL SCOPE]
          [COND
             (STATEMENT (SETQ ANSWER `(,@ANSWER finally ,@(BUILD.TAIL STATEMENT]
          [COND
             (EXITLIST (SETQ ANSWER (BUILD.PROGN (CONS ANSWER EXITLIST]
          (RETURN ANSWER])
```

(PARSE.STATEMENT.RETURN

```
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (OPTARGS ANSWER)                                       (* (statement RETURN optargs) *)
          (PARSE.BIN 'RETURN)
          (SETQ OPTARGS (PARSE.OPTARGS))
          (SETQ ANSWER (BUILD.RETURN OPTARGS))
          (RETURN ANSWER])
```

(PARSE.STATEMENT.TRANSFER

```
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (TRANSFER LHS ANSWER)                                  (* (statement transfer lhs) *)
          (SETQ TRANSFER (PARSE.TRANSFER))
          (SETQ LHS (PARSE.LHS))
          [SETQ ANSWER `(SHOULDNT ',LHS]
          (RETURN ANSWER])
```

(PARSE.STATEMENT.LBRACKET

```
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (EXPLIST EXP ANSWER)                                   (* (statement %[ explist %] _ exp) *)
          (PARSE.BIN '%[)
          (SETQ EXPLIST (PARSE.EXPLIST))
          (PARSE.BIN '%])
          (PARSE.BIN '_)
          (SETQ EXP (PARSE.EXP))
          (SETQ ANSWER (BUILD.SETQ EXPLIST EXP))
          (RETURN ANSWER])
```

```
(PARSE.STATEMENT.IF
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:47")
    (PROG (EXP STATEMENT ELSEPART HEAD TAIL ANSWER)        (* (statement IF exp THEN statement elsepart) *)
          (PARSE.BIN 'IF)
          (SETQ EXP (PARSE.EXP))
          (PARSE.BIN 'THEN)
          (SETQ STATEMENT (PARSE.STATEMENT))
          (SETQ ELSEPART (PARSE.ELSEPART))
          (SETQ ANSWER (BUILD.COND EXP STATEMENT ELSEPART))
          (RETURN ANSWER])

(PARSE.BLOCK
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:47")
    (PROG (SCOPE EXITS ANSWER)                             (* (block BEGIN scope exits END)
                                                           (block { scope exits }) *)
          (BUILD.PUSH.SCOPE)
          (PARSE.BIN '(BEGIN {))
          (SETQ SCOPE (PARSE.SCOPE))
          (SETQ EXITS (PARSE.EXITS))
          (PARSE.BIN '(END }))
          (BUILD.POP.SCOPE)
          (SETQ ANSWER (APPEND SCOPE EXITS))
          (RETURN ANSWER])

(PARSE.SCOPE
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:47")
    (PROG (STATEMENTLIST ANSWER)                           (* (scope open enables statementlist)
                                                           (scope open enables declist ; statementlist) *)
          (BUILD.STORE.OPEN (PARSE.OPEN))
          (PARSE.ENABLES)
          (COND
             ([AND (EQ PARSE.CLASS 'ID)
                   (FMEMB PARSE.ATOM2 '(%, %:]
              (PARSE.DECLIST)))
          (SETQ STATEMENTLIST (PARSE.STATEMENTLIST))
          (SETQ ANSWER (BUILD.PROG STATEMENTLIST))
          (RETURN ANSWER])

(PARSE.BINDITEM
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:47")
    (PROG (ID EXP ANSWER)                                  (* BOTH (binditem exp) (binditem id %: exp) *)
                                                           (* CEDAR (binditem id ~ ~ exp) *)
          [COND
             ((AND (EQ PARSE.CLASS 'ID)
                   (EQ PARSE.ATOM2 '%:))
              (SETQ ID (PARSE.BIN))
              (PARSE.BIN))
             ((AND (EQ PARSE.LANGUAGE 'CEDAR)
                   (EQ PARSE.CLASS 'ID)
                   (EQ PARSE.ATOM2 '~))
              (SETQ ID (PARSE.BIN))
              (PARSE.BIN)
              (PARSE.BIN '~]
          (SETQ EXP (PARSE.EXP))
          (SETQ ANSWER (create BINDITEM
                               ID _ ID
                               EXP _ EXP))
          (RETURN ANSWER])

(PARSE.EXITS
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:47")
                                                           (* (exits EXITS exitlist) (exits) *)
    (COND
       ((EQ PARSE.ATOM 'EXITS)
        (PARSE.BIN)
        (PARSE.EXITLIST])

(PARSE.CASESTMTITEM
  [LAMBDA (CASEHEAD)                                       (* kbr%: "25-Nov-85 12:47")
    (PROG (CASELABEL STATEMENT ANSWER)                     (* (casestmtitem caselabel => statement) *)
          (SETQ CASELABEL (PARSE.CASELABEL))
          (PARSE.BIN '=>)
          (SETQ STATEMENT (PARSE.STATEMENT))
          (SETQ ANSWER (CONS CASELABEL (BUILD.TAIL STATEMENT)))
          (RETURN ANSWER])

(PARSE.CASEEXPITEM
  [LAMBDA (CASEHEAD)                                       (* kbr%: "25-Nov-85 12:47")
    (PROG (CASELABEL EXP ANSWER)                           (* (caseexpitem caselabel => exp) *)
          (SETQ CASELABEL (PARSE.CASELABEL))
          (PARSE.BIN '=>)
          (SETQ EXP (PARSE.EXP))
          (SETQ ANSWER (CONS CASELABEL (BUILD.TAIL EXP)))
          (RETURN ANSWER])
```

(PARSE.EXITITEM

```
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
    (PROG (IDLIST STATEMENT ANSWER)                               (* (exititem idlist => statement) *)
          (SETQ IDLIST (PARSE.IDLIST))
          (PARSE.BIN '=>)
          (SETQ STATEMENT (PARSE.STATEMENT))
          [SETQ ANSWER (BUILD.PROGN (NCONC IDLIST (BUILD.TAIL STATEMENT]
          (RETURN ANSWER])
```

(PARSE.CASETEST

```
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
    (PROG (OPTRELATION EXP ANSWER)                                (* (casetest optrelation) (casetest exp) *)
          (COND
             ((FMEMB PARSE.ATOM PARSE.OPTRELATION.FIRST)
              (SETQ OPTRELATION (PARSE.OPTRELATION))
              (SETQ ANSWER OPTRELATION))
             (T (SETQ EXP (PARSE.EXP))
                (SETQ ANSWER EXP)))
          (RETURN ANSWER])
```

(PARSE.CONTROLID

```
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
    (PROG (ID TYPEEXP)                                            (* (controlid ident typeexp) (controlid id) *)
          [COND
             ((FMEMB PARSE.ATOM2 PARSE.CONTROLID.FOLLOW)
              (SETQ ID (PARSE.BIN 'ID))
              (SETQ TYPEEXP 'INTEGER))
             (T (SETQ ID (PARSE.IDENT))
                (SETQ TYPEEXP (PARSE.TYPEEXP]
          (BUILD.INITIALIZE.VAR ID TYPEEXP NIL BUILD.CURRENT.SCOPE)
          (RETURN ID])
```

(PARSE.FORCLAUSE

```
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
    (PROG (CONTROLID EXP1 EXP2 DIRECTION RANGE ANSWER)            (* (forclause FOR controlid _ exp %, exp)
                                                                  (forclause FOR controlid direction IN range)
                                                                  (forclause THROUGH range) (forclause) *)
          [COND
             [(EQ PARSE.ATOM 'FOR)
              (PARSE.BIN)
              (SETQ CONTROLID (PARSE.CONTROLID))
              (COND
                 ((EQ PARSE.ATOM '_)
                  (PARSE.BIN)
                  (SETQ EXP1 (PARSE.EXP))
                  (PARSE.BIN '%,)
                  (SETQ EXP2 (PARSE.EXP))
                  (SETQ ANSWER (BUILD.FORCLAUSE.BY CONTROLID EXP1 EXP2)))
                 (T (SETQ DIRECTION (PARSE.DIRECTION))
                    (PARSE.BIN 'IN)
                    (SETQ RANGE (PARSE.RANGE))
                    (SETQ ANSWER (BUILD.FORCLAUSE.IN CONTROLID DIRECTION RANGE]
             ((EQ PARSE.ATOM 'THROUGH)
              (PARSE.BIN)
              (SETQ RANGE (PARSE.RANGE))
              (SETQ ANSWER (BUILD.FORCLAUSE.THROUGH RANGE]
          (RETURN ANSWER])
```

(PARSE.DIRECTION

```
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
                                                                  (* (direction DECREASING) (direction) *)
    (COND
       ((EQ PARSE.ATOM 'DECREASING)
        (PARSE.BIN])
```

(PARSE.DOTEST

```
  [LAMBDA NIL                                                     (* kbr%: "25-Nov-85 12:47")
                                                                  (* (dotest UNTIL exp) (dotest WHILE exp)
                                                                  (dotest) *)
    (COND
       ((EQ PARSE.ATOM 'UNTIL)
        (PARSE.BIN)
        (LIST 'until (PARSE.EXP)))
       ((EQ PARSE.ATOM 'WHILE)
        (PARSE.BIN)
        (LIST 'while (PARSE.EXP])
(PARSE.DOEXIT
  [LAMBDA NIL                                          (* kbr%: "25-Nov-85 12:47")
    (PROG (EXITLIST STATEMENT ANSWER)                  (* (doexit) (doexit REPEAT exitlist)
                                                       (doexit REPEAT exitlist FINISHED => statement)
                                                       (doexit REPEAT exitlist FINISHED => statement ;) *)
          [COND
             ((EQ PARSE.ATOM 'REPEAT)
              (PARSE.BIN)
              (SETQ EXITLIST (PARSE.EXITLIST))
              (COND
                 ((EQ PARSE.ATOM 'FINISHED)
                  (PARSE.BIN)
                  (PARSE.BIN '=>)
                  (SETQ STATEMENT (PARSE.STATEMENT))
                  (COND
                     ((EQ PARSE.ATOM ';)
                      (PARSE.BIN]
          (SETQ ANSWER (LIST EXITLIST STATEMENT))
          (RETURN ANSWER])

(PARSE.ENABLES
  [LAMBDA NIL                                          (* kbr%: "25-Nov-85 12:47")
    (PROG (CATCHCASE CATCHANY CATCHLIST ANSWER)        (* (enables ENABLE catchcase ;)
                                                       (enables ENABLE catchany ;) (enables ENABLE BEGIN
                                                       catchlist END ;) (enables ENABLE { catchlist } ;)
                                                       (enables) *)
          [COND
             ((EQ PARSE.ATOM 'ENABLE)
              (PARSE.BIN)
              [COND
                 ((EQ PARSE.ATOM 'ANY)
                  (SETQ CATCHANY (PARSE.CATCHANY))
                  (SETQ ANSWER (LIST CATCHANY)))
                 ((FMEMB PARSE.ATOM '(BEGIN {))
                  (PARSE.BIN)
                  (SETQ CATCHLIST (PARSE.CATCHLIST))
                  (PARSE.BIN '(END }))
                  (SETQ ANSWER CATCHLIST))
                 (T (SETQ CATCHCASE (PARSE.CATCHCASE))
                    (SETQ ANSWER (LIST CATCHCASE]
              (PARSE.BIN ';]
          (RETURN ANSWER])

(PARSE.CATCHLIST
  [LAMBDA NIL                                          (* kbr%: "25-Nov-85 12:47")
    (PROG (CATCHHEAD CATCHANY CATCHCASE ANSWER)        (* (catchlist catchhead) (catchlist catchhead catchcase)
                                                       (catchlist catchhead catchany) (catchlist catchhead catchany ;) *)
          (SETQ CATCHHEAD (PARSE.CATCHHEAD))
          [COND
             ((FMEMB PARSE.ATOM PARSE.CATCHLIST.FOLLOW)
              (SETQ ANSWER CATCHHEAD))
             [(EQ PARSE.ATOM 'ANY)
              (SETQ CATCHANY (PARSE.CATCHANY))
              (SETQ ANSWER (NCONC1 CATCHHEAD CATCHANY))
              (COND
                 ((EQ PARSE.ATOM ';)
                  (PARSE.BIN]
             (T (SETQ CATCHCASE (PARSE.CATCHCASE))
                (SETQ ANSWER (NCONC1 CATCHHEAD CATCHCASE]
          (RETURN ANSWER])

(PARSE.CATCHCASE
  [LAMBDA NIL                                          (* kbr%: "25-Nov-85 12:47")
    (PROG (LHSLIST STATEMENT ANSWER)                   (* (catchcase lhslist => statement) *)
          (SETQ LHSLIST (PARSE.LHSLIST))
          (PARSE.BIN '=>)
          (SETQ STATEMENT (PARSE.STATEMENT))
          (SETQ ANSWER (LIST 'catchcase LHSLIST STATEMENT))
          (RETURN ANSWER])

(PARSE.OPTARGS
  [LAMBDA NIL                                          (* kbr%: "25-Nov-85 12:47")
    (PROG (ANSWER)                                     (* (optargs %[ explist %]) (optargs)
                                                       (optargs lhs) *)
          [COND
             ((EQ PARSE.ATOM '%[)
              (PARSE.BIN '%[)
              (SETQ ANSWER (PARSE.EXPLIST))
              (PARSE.BIN '%]))
             ((NOT (FMEMB PARSE.ATOM PARSE.OPTARGS.FOLLOW))
              (SETQ ANSWER (LIST (PARSE.LHS]
          (RETURN ANSWER])

(PARSE.TRANSFER
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
                                                                 (* (transfer SIGNAL) (transfer ERROR)
                                                                 (transfer RETURN WITH ERROR)
                                                                 (transfer START) (transfer RESTART)
                                                                 (transfer JOIN) (transfer NOTIFY)
                                                                 (transfer BROADCAST) (transfer TRANSFER WITH)
                                                                 (transfer RETURN WITH) *)
    (COND
       [(EQ PARSE.ATOM 'RETURN)
        (PARSE.BIN)
        (PARSE.BIN 'WITH)
        (COND
           ((EQ PARSE.ATOM 'ERROR)
            'SHOULDNT)
           (T 'RETURN]
       ((EQ PARSE.ATOM 'TRANSFER)
        (PARSE.BIN)
        (PARSE.BIN 'WITH)
        'RETURN)
       (T (PARSE.BIN])

(PARSE.KEYITEM
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (ID OPTEXP ANSWER)                                     (* BOTH (keyitem id %: optexp) *)
                                                                 (* CEDAR (keyitem id ~ optexp) *)
          (SETQ ID (PARSE.BIN 'ID))
          (COND
             ((EQ PARSE.ATOM '%:)
              (PARSE.BIN))
             ((AND (EQ PARSE.LANGUAGE 'CEDAR)
                   (EQ PARSE.ATOM '~))
              (PARSE.BIN))
             (T (SHOULDNT)))
          (SETQ OPTEXP (PARSE.OPTEXP))
          (SETQ ANSWER (create KEYITEM
                               ID _ ID
                               OPTEXP _ OPTEXP))
          (RETURN ANSWER])

(PARSE.OPTEXP
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (ANSWER)                                               (* (optexp TRASH) (optexp NULL)
                                                                 (optexp exp) (optexp) *)
          [SETQ ANSWER (COND
                          ((FMEMB PARSE.ATOM '(NULL TRASH))
                           (PARSE.BIN)
                           'TRASH)
                          ((FMEMB PARSE.ATOM PARSE.OPTEXP.FOLLOW)
                           'TRASH)
                          (T (PARSE.EXP]
          (RETURN ANSWER])

(PARSE.EXP
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (COND
       ((EQ PARSE.CLASS 'ID)
        (PARSE.EXP1))
       (T (PARSE.EXP2])

(PARSE.EXP1
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
                                                                 (* First token of EXP is ID. *)
    (PROG (DISJUNCT EXP ANSWER)                                  (* (exp lhs _ exp) (exp disjunct) *)
          (SETQ DISJUNCT (PARSE.DISJUNCT))
          (COND
             ((EQ PARSE.ATOM '_)
              (PARSE.BIN)
              (SETQ EXP (PARSE.EXP))
              (SETQ ANSWER (BUILD.SETQ DISJUNCT EXP)))
             (T (SETQ ANSWER DISJUNCT)))
          (RETURN ANSWER])

(PARSE.EXP2
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
                                                                 (* First token of EXP is not ID. *)
    (PROG (DISJUNCT ANSWER)                                      (* (exp transferop lhs) (exp IF exp THEN exp ELSE exp)
                                                                 (exp casehead caseexplist ENDCASE => exp)
                                                                 (exp lhs _ exp) (exp %[ explist %] _ exp)
                                                                 (exp ERROR) (exp disjunct) *)
          [SETQ ANSWER
           (COND
              ([AND (FMEMB PARSE.ATOM PARSE.TRANSFEROP.FIRST)
                    (OR (NOT (EQ PARSE.ATOM 'NEW))
                        (NOT (EQ PARSE.ATOM2 '%[]           (* Don't confuse with (primary new %[ typeexp initialization
                                                              optcatch %]) *)
               (PARSE.EXP.TRANSFEROP))
              ((EQ PARSE.ATOM 'IF)
               (PARSE.EXP.IF))
              ((FMEMB PARSE.ATOM PARSE.CASEHEAD.FIRST)
               (PARSE.EXP.CASEHEAD))
              ((EQ PARSE.ATOM)
               (PARSE.EXP.LBRACKET '%[))
              ((EQ PARSE.ATOM 'ERROR)
               (PARSE.EXP.ERROR))
              ((NUMBERP PARSE.ATOM)
               (PARSE.EXP.DISJUNCT))
              ((STRINGP PARSE.ATOM)
               (PARSE.EXP.DISJUNCT))
              ((FMEMB PARSE.ATOM
                      '(ABS ALL BASE DESCRIPTOR FIRST ISTYPE LAST LENGTH LONG MAX MIN NILL NOT ORD PRED SIZE
                            SUCC VAL + - @ %[))
               (PARSE.EXP.DISJUNCT))
              (T (PROGN (SETQ DISJUNCT (PARSE.EXP.DISJUNCT))
                        (COND
                           ((EQ PARSE.ATOM '_)
                            (PARSE.BIN)
                            (BUILD.SETQ DISJUNCT (PARSE.EXP)))
                           (T DISJUNCT]
          (RETURN ANSWER])
```

(PARSE.EXP.TRANSFEROP

```
  [LAMBDA NIL                                                 (* kbr%: "25-Nov-85 12:47")
    (PROG (TRANSFEROP LHS ANSWER)                             (* (exp transferop lhs) *)
          (SETQ TRANSFEROP (PARSE.TRANSFEROP))
          (SETQ LHS (PARSE.LHS))
          [SETQ ANSWER `(SHOULDNT ',LHS]
          (RETURN ANSWER])
```

(PARSE.EXP.IF

```
  [LAMBDA NIL                                                 (* kbr%: "25-Nov-85 12:47")
    (PROG (EXP1 EXP2 EXP3 ANSWER)                             (* (exp IF exp THEN exp ELSE exp) *)
          (PARSE.BIN 'IF)
          (SETQ EXP1 (PARSE.EXP))
          (PARSE.BIN 'THEN)
          (SETQ EXP2 (PARSE.EXP))
          (PARSE.BIN 'ELSE)
          (SETQ EXP3 (PARSE.EXP))
          (SETQ ANSWER (BUILD.COND EXP1 EXP2 EXP3))
          (RETURN ANSWER])
```

(PARSE.EXP.CASEHEAD

```
  [LAMBDA NIL                                                 (* kbr%: "25-Nov-85 12:47")
    (PROG (CASEHEAD CASEEXPLIST EXP ANSWER)                   (* (exp casehead caseexplist ENDCASE => exp) *)
          (SETQ CASEHEAD (PARSE.CASEHEAD))
          (SETQ CASEEXPLIST (PARSE.CASEEXPLIST))
          (PARSE.BIN 'ENDCASE)
          (PARSE.BIN '=>)
          (SETQ EXP (PARSE.EXP))
          (SETQ ANSWER (BUILD.SELECTQ CASEHEAD CASEEXPLIST EXP))
          (RETURN ANSWER])
```

(PARSE.EXP.LHS

```
  [LAMBDA NIL                                                 (* kbr%: "25-Nov-85 12:47")
    (PROG (LHS EXP ANSWER)                                    (* (exp lhs _ exp) *)
          (SETQ LHS (PARSE.LHS))
          (PARSE.BIN '_)
          (SETQ EXP (PARSE.EXP))
          [SETQ ANSWER `(SETQ ,LHS ,EXP]
          (RETURN ANSWER])
```

(PARSE.EXP.LBRACKET

```
  [LAMBDA NIL                                                 (* kbr%: "25-Nov-85 12:47")
    (PROG (EXPLIST EXP ANSWER)                                (* (exp %[ explist %] _ exp) *)
          (PARSE.BIN '%[)
          (SETQ EXPLIST (PARSE.EXPLIST))
          (PARSE.BIN '%])
          (PARSE.BIN '_)
          (SETQ EXP (PARSE.EXP))
          [SETQ ANSWER `(SETQ ,EXPLIST ,EXP]
          (RETURN ANSWER])
```

```
(PARSE.EXP.ERROR
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
                                                                 (* (exp ERROR) *)
    (PARSE.BIN 'ERROR)
    '(SHOULDNT])

(PARSE.EXP.DISJUNCT
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
                                                                 (* (exp disjunct) *)
    (PARSE.DISJUNCT])

(PARSE.DISJUNCT
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (CONJUNCTS ANSWER)                                     (* (disjunct disjunct OR conjunct)
                                                                 (disjunct conjunct) *)
          (push CONJUNCTS (PARSE.CONJUNCT))
          (while (EQ PARSE.ATOM 'OR) do (PARSE.BIN)
                                        (push CONJUNCTS (PARSE.CONJUNCT)))
          [SETQ ANSWER (COND
                          ((CDR CONJUNCTS)
                           (CONS 'OR (DREVERSE CONJUNCTS)))
                          (T (CAR CONJUNCTS]
          (RETURN ANSWER])

(PARSE.CONJUNCT
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (NEGATIONS ANSWER)                                     (* (conjunct conjunct AND negation)
                                                                 (conjunct negation) *)
          (push NEGATIONS (PARSE.NEGATION))
          (while (EQ PARSE.ATOM 'AND) do (PARSE.BIN)
                                         (push NEGATIONS (PARSE.NEGATION)))
          [SETQ ANSWER (COND
                          ((CDR NEGATIONS)
                           (CONS 'AND (DREVERSE NEGATIONS)))
                          (T (CAR NEGATIONS]
          (RETURN ANSWER])

(PARSE.NEGATION
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (NOT ANSWER)                                           (* (negation not relation) (negation relation) *)
          [COND
             ((FMEMB PARSE.ATOM PARSE.NOTS)
              (SETQ NOT (PARSE.NOT]
          (SETQ ANSWER (PARSE.RELATION))
          [COND
             (NOT (SETQ ANSWER `(NOT ,ANSWER]
          (RETURN ANSWER])

(PARSE.RELATION
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (SUM OPTRELATION NOT ANSWER)                           (* (relation sum optrelation) (relation sum) *)
          (SETQ SUM (PARSE.SUM))
          (COND
             ((NOT (FMEMB PARSE.ATOM PARSE.OPTRELATION.FIRST))
              (RETURN SUM)))
          (SETQ OPTRELATION (PARSE.OPTRELATION))
          [COND
             ((EQ (CAR OPTRELATION)
                  'NOT)
              (SETQ NOT T)
              (SETQ OPTRELATION (CADR OPTRELATION]
          [SETQ ANSWER (COND
                          ((EQ (CAR OPTRELATION)
                               'IN)
                           (BUILD.IN SUM (CADR OPTRELATION)))
                          (T (BUILD.ARITH.EXP2 (CAR OPTRELATION)
                                      SUM
                                      (CADR OPTRELATION]
          [COND
             (NOT (SETQ ANSWER (LIST 'NOT ANSWER]
          (RETURN ANSWER])

(PARSE.SUM
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:47")
    (PROG (PRODUCTS PRODUCT ANSWER)                              (* (sum sum addop product) (sum product) *)
          (SETQ PRODUCT (PARSE.PRODUCT))
          [while (FMEMB PARSE.ATOM PARSE.ADDOPS) do (COND
                                                       ((EQ PARSE.ATOM '+)
                                                        (PARSE.BIN)
                                                        (push PRODUCTS PRODUCT)
```

```
                                                               (SETQ PRODUCT (PARSE.PRODUCT)))
                                                              [(EQ PARSE.ATOM '-)
                                                               (PARSE.BIN)
                                                               (SETQ PRODUCT (BUILD.ARITH.EXP2 '- PRODUCT (
                                                                                                    PARSE.PRODUCT
                                                                                                     ]
                                                              (T (SHOULDNT]
          (push PRODUCTS PRODUCT)
          [SETQ ANSWER (COND
                          ((CDR PRODUCTS)
                           (BUILD.ARITH.EXP* '+ (DREVERSE PRODUCTS)))
                          (T (CAR PRODUCTS]
          (RETURN ANSWER])
```

```
(PARSE.PRODUCT
  [LAMBDA NIL                                                  (* kbr%: "25-Nov-85 12:47")
    (PROG (FACTORS FACTOR ANSWER)                              (* (product product multop factor)
                                                               (product factor) *)
          (SETQ FACTOR (PARSE.FACTOR))
          [while (FMEMB PARSE.ATOM PARSE.MULTOPS) do (COND
                                                               ((EQ PARSE.ATOM '*)
                                                                (PARSE.BIN)
                                                                (push FACTORS FACTOR)
                                                                (SETQ FACTOR (PARSE.FACTOR)))
                                                               [(EQ PARSE.ATOM '/)
                                                                (PARSE.BIN)
                                                                (SETQ FACTOR (BUILD.ARITH.EXP2 '/ FACTOR (PARSE.FACTOR]
                                                               [(EQ PARSE.ATOM 'MOD)
                                                                (PARSE.BIN)
                                                                (SETQ FACTOR (BUILD.ARITH.EXP2 'MOD FACTOR (
                                                                                                     PARSE.FACTOR
                                                                                                      ]
                                                               (T (SHOULDNT]
          (push FACTORS FACTOR)
          [SETQ ANSWER (COND
                          ((CDR FACTORS)
                           (BUILD.ARITH.EXP* '* (DREVERSE FACTORS)))
                          (T (CAR FACTORS]
          (RETURN ANSWER])
```

```
(PARSE.OPTRELATION
  [LAMBDA NIL                                                  (* kbr%: "25-Nov-85 12:47")
    (PROG (NOT ANSWER)                                         (* (optrelation not relationtail) (optrelation relationtail) *)
                                                               (* In CEDAR, not must be NOT. *)
          [COND
             ([OR (EQ PARSE.ATOM 'NOT)
                  (AND (EQ PARSE.LANGUAGE 'MESA)
                       (EQ PARSE.ATOM '~]
              (SETQ NOT (PARSE.NOT]
          (SETQ ANSWER (PARSE.RELATIONTAIL))
          [COND
             (NOT (SETQ ANSWER (LIST 'NOT ANSWER]
          (RETURN ANSWER])
```

```
(PARSE.RELATIONTAIL
  [LAMBDA NIL                                                  (* kbr%: "25-Nov-85 12:47")
    (PROG (RANGE RELOP SUM ANSWER)                             (* (relationtail IN range) (relationtail relop sum) *)
          [COND
             ((EQ PARSE.ATOM 'IN)
              (PARSE.BIN)
              (SETQ RANGE (PARSE.RANGE))
              (SETQ ANSWER (LIST 'IN RANGE)))
             (T (SETQ RELOP (PARSE.RELOP))
                (SETQ SUM (PARSE.SUM))
                (SETQ ANSWER (LIST RELOP SUM]
          (RETURN ANSWER])
```

```
(PARSE.RELOP
  [LAMBDA NIL                                                  (* kbr%: "25-Nov-85 12:47")
                                                               (* (relop =) (relop %#) (relop <) (relop <=)
                                                               (relop >) (relop >=) *)
    (PARSE.BIN])
```

```
(PARSE.ADDOP
  [LAMBDA NIL                                                  (* kbr%: "25-Nov-85 12:47")
                                                               (* (addop +) (addop -) *)
    (PARSE.BIN])
```

```
(PARSE.MULTOP
  [LAMBDA NIL                                                  (* kbr%: "25-Nov-85 12:48")
                                                               (* (multop *) (multop /) (multop MOD) *)
```

```
    (PARSE.BIN])

(PARSE.FACTOR
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 12:48")
    (PROG (ADDOP PRIMARY ANSWER)                                     (* (factor addop primary) (factor primary) *)
          [COND
             ((FMEMB PARSE.ATOM PARSE.ADDOPS)
              (SETQ ADDOP (PARSE.BIN]
          (SETQ ANSWER (PARSE.PRIMARY))
          [COND
             ((EQ ADDOP '-)
              (SETQ ANSWER (BUILD.ARITH.EXP1 '- ANSWER]
          (RETURN ANSWER])

(PARSE.PRIMARY
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 12:48")
    (PROG (ANSWER LHS)

          (* BOTH (primary num) (primary lnum) (primary flnum) (primary string)
          (primary lstring) (primary atom) (primary NIL) (primary %[ explist %])
          (primary prefixop %[ orderlist %]) (primary VAL %[ orderlist %])
          (primary ALL %[ orderlist %]) (primary new %[ typeexp initialization optcatch %])
          (primary typeop %[ typeexp %]) (primary SIZE %[ typeexp %]) (primary SIZE %[ typeexp %, exp %])
          (primary ISTYPE %[ exp %, typeexp %]) (primary @ lhs) (primary DESCRIPTOR %[ desclist %])
          (primary lhs) *)
                                                                     (* CEDAR (primary cons %[ explist optcatch %])
                                                                     (primary listcons %[ explist %]) *)
                                                                     (* In CEDAR, new can be NEW. *)
          [SETQ ANSWER (COND
                          ((EQ PARSE.CLASS 'CHAR)
                           (BUILD.CHARCODE (PARSE.BIN)))
                          ((NUMBERP PARSE.ATOM)
                           (PARSE.BIN))
                          ((STRINGP PARSE.ATOM)
                           (PARSE.BIN))
                          ((FMEMB PARSE.ATOM PARSE.PREFIXOP.FIRST)
                           (PARSE.PRIMARY.PREFIXOP))
                          ((AND [OR (FMEMB PARSE.ATOM PARSE.TYPEOP.FIRST)
                                    (AND (EQ PARSE.LANGUAGE 'CEDAR)
                                         (EQ PARSE.ATOM 'CODE]
                                (EQ PARSE.ATOM2 '%[))
                           (PARSE.PRIMARY.TYPEOP))
                          (T (SELECTQ PARSE.ATOM
                                 ($ (PARSE.ATOM))
                                 (NILL (PARSE.PRIMARY.NIL))
                                 (%[ (PARSE.PRIMARY.LBRACKET))
                                 (VAL (PARSE.PRIMARY.VAL))
                                 (ALL (PARSE.PRIMARY.ALL))
                                 (SIZE (PARSE.PRIMARY.SIZE))
                                 (ISTYPE (PARSE.PRIMARY.ISTYPE))
                                 (@ (PARSE.PRIMARY.AT))
                                 (DESCRIPTOR (PARSE.PRIMARY.DESCRIPTOR))
                                 (NEW (PARSE.PRIMARY.NEW))
                                 (CONS (PARSE.PRIMARY.CONS))
                                 (LIST (PARSE.PRIMARY.LIST))
                                 (PARSE.PRIMARY.LHS]
          (RETURN ANSWER])

(PARSE.ATOM
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 12:48")
                                                                     (* (atom $ id) *)
    (PARSE.BIN '$)
    `',(PARSE.BIN 'ID])

(PARSE.PRIMARY.NIL
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 12:48")
                                                                     (* (primary NIL) *)
    (PARSE.BIN 'NILL)
    NIL])

(PARSE.PRIMARY.LBRACKET
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 12:48")
    (PROG (ANSWER)                                                   (* (primary %[ explist %]) *)
          (PARSE.BIN '%[)
          (SETQ ANSWER (PARSE.EXPLIST))
          (PARSE.BIN '%])
          (RETURN ANSWER])

(PARSE.PRIMARY.PREFIXOP
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 12:48")
    (PROG (PREFIXOP ORDERLIST ANSWER)                                (* (primary prefixop %[ orderlist %]) *)
          (SETQ PREFIXOP (PARSE.PREFIXOP))
          (PARSE.BIN '%[)
          (SETQ ORDERLIST (PARSE.ORDERLIST))
          (PARSE.BIN '%])
          (SETQ ANSWER (CONS PREFIXOP (fetch (ORDERLIST ITEMS) of ORDERLIST)))
          (RETURN ANSWER])
```

(PARSE.PRIMARY.VAL

```
  [LAMBDA NIL                                        (* kbr%: "25-Nov-85 12:48")
    (PROG (ORDERLIST ANSWER)                         (* (primary VAL %[ orderlist %]) *)
          (PARSE.BIN 'VAL)
          (PARSE.BIN '%[)
          (SETQ ORDERLIST (PARSE.ORDERLIST))
          (PARSE.BIN '%])
          (SETQ ANSWER (CONS 'VAL (fetch (ORDERLIST ITEMS) of ORDERLIST)))
          (RETURN ANSWER])
```

(PARSE.PRIMARY.ALL

```
  [LAMBDA NIL                                        (* kbr%: "25-Nov-85 12:48")
    (PROG (ORDERLIST ANSWER)                         (* (primary ALL %[ orderlist %]) *)
          (PARSE.BIN 'ALL)
          (PARSE.BIN '%[)
          (SETQ ORDERLIST (PARSE.ORDERLIST))
          (PARSE.BIN '%])
          (SETQ ANSWER (CONS 'ALL (fetch (ORDERLIST ITEMS) of ORDERLIST)))
          (RETURN ANSWER])
```

(PARSE.PRIMARY.NEW

```
  [LAMBDA NIL                                        (* kbr%: "25-Nov-85 12:48")
    (PROG (TYPEEXP INITIALIZATION ANSWER)            (* (primary new %[ typeexp initialization optcatch %]) *)
          (PARSE.NEW)
          (PARSE.BIN '%[)
          (SETQ TYPEEXP (PARSE.TYPEEXP))
          (SETQ INITIALIZATION (PARSE.INITIALIZATION))
          (PARSE.OPTCATCH)
          (PARSE.BIN '%])
          (SETQ ANSWER (BUILD.NEW TYPEEXP INITIALIZATION))
          (RETURN ANSWER])
```

(PARSE.PRIMARY.TYPEOP

```
  [LAMBDA NIL                                        (* kbr%: "25-Nov-85 12:48")
    (PROG (TYPEOP TYPEEXP ANSWER)                    (* (primary typeop %[ typeexp %]) *)
          (SETQ TYPEOP (PARSE.TYPEOP))
          (PARSE.BIN '%[)
          (SETQ TYPEEXP (PARSE.TYPEEXP))
          (PARSE.BIN '%])
          (SETQ ANSWER (LIST TYPEOP TYPEEXP))
          (RETURN ANSWER])
```

(PARSE.PRIMARY.SIZE

```
  [LAMBDA NIL                                        (* kbr%: "25-Nov-85 12:48")
    (PROG (TYPEEXP EXP ANSWER)                       (* (primary SIZE %[ typeexp %]) (primary SIZE %[ typeexp %,
                                                     exp %]) *)

          (PARSE.BIN 'SIZE)
          (PARSE.BIN '%[)
          (SETQ TYPEEXP (PARSE.TYPEEXP))
          [COND
             ((NOT (EQ PARSE.ATOM '%,))
              (PARSE.BIN '%])
              (SETQ ANSWER (LIST 'SIZE TYPEEXP)))
             (T (PARSE.BIN)
                (SETQ EXP (PARSE.EXP))
                (PARSE.BIN '%])
                (SETQ ANSWER (LIST 'SIZE TYPEEXP EXP]
          (RETURN ANSWER])
```

(PARSE.PRIMARY.ISTYPE

```
  [LAMBDA NIL                                        (* kbr%: "25-Nov-85 12:48")
                                                     (* (primary ISTYPE %[ exp %, typeexp %]) *)

    (PROG (EXP TYPEEXP ANSWER)
          (PARSE.BIN 'ISTYPE)
          (PARSE.BIN '%[)
          (SETQ EXP (PARSE.EXP))
          (PARSE.BIN '%,)
          (SETQ TYPEEXP (PARSE.TYPEEXP))
          (PARSE.BIN '%])
          (SETQ ANSWER (BUILD.ISTYPE EXP TYPEEXP))
          (RETURN ANSWER])
```

(PARSE.PRIMARY.AT

```
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
    (PROG (LHS ANSWER)                                     (* (primary @ lhs) *)
          (PARSE.BIN '@)
          (SETQ LHS (PARSE.LHS))
          (SETQ ANSWER LHS)
          (RETURN ANSWER])
```

(PARSE.PRIMARY.DESCRIPTOR

```
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
    (PROG (DESCLIST ANSWER)                                (* (primary DESCRIPTOR %[ desclist %]) *)
          (PARSE.BIN 'DESCRIPTOR)
          (PARSE.BIN '%[)
          (SETQ DESCLIST (PARSE.DESCLIST))
          (PARSE.BIN '%])
          (SETQ ANSWER (CONS 'DESCRIPTOR DESCLIST))
          (RETURN ANSWER])
```

(PARSE.PRIMARY.CONS

```
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
    (PROG (EXPLIST ANSWER)                                 (* CEDAR (primary CONS %[ explist optcatch %]) *)
          (PARSE.THISIS.CEDAR)
          (PARSE.BIN 'CONS)
          (PARSE.BIN '%[)
          (SETQ EXPLIST (PARSE.EXPLIST))
          (PARSE.OPTCATCH)
          (PARSE.BIN '%])
          (SETQ ANSWER (CONS 'CONS (fetch (EXPLIST ITEMS) of EXPLIST)))
          (RETURN ANSWER])
```

(PARSE.PRIMARY.LIST

```
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
    (PROG (EXPLIST ANSWER)                                 (* CEDAR (primary LIST %[ explist %]) *)
          (PARSE.THISIS.CEDAR)
          (PARSE.BIN 'LIST)
          (PARSE.BIN '%[)
          (SETQ EXPLIST (PARSE.EXPLIST))
          (PARSE.BIN '%])
          (SETQ ANSWER (CONS 'LIST (fetch (EXPLIST ITEMS) of EXPLIST)))
          (RETURN ANSWER])
```

(PARSE.PRIMARY.LHS

```
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
    (PROG (LHS QUALIFIER ANSWER)                           (* BOTH (primary lhs) (primary new %[ typeexp initialization
                                                           optcatch %]) *)
                                                           (* CEDAR (primary cons %[ explist optcatch %])
                                                           (primary listcons %[ explist %]) *)
          (SETQ LHS (PARSE.LHS))
          (COND
             ([NOT (AND (EQ PARSE.ATOM '%.)
                        (OR (EQ PARSE.ATOM2 'NEW)
                            (AND (EQ PARSE.LANGUAGE 'CEDAR)
                                 (FMEMB PARSE.ATOM2 '(CONS LIST]
              (RETURN LHS)))
          (PARSE.BIN '%.)
          (SETQ ANSWER (SELECTQ PARSE.ATOM
                           (NEW (PARSE.PRIMARY.LHS.NEW LHS))
                           (CONS (PARSE.PRIMARY.LHS.CONS LHS))
                           (LIST (PARSE.PRIMARY.LHS.LIST LHS))
                           (SHOULDNT)))
          (RETURN ANSWER])
```

(PARSE.PRIMARY.LHS.NEW

```
  [LAMBDA (LHS)                                            (* kbr%: "25-Nov-85 12:48")
    (PROG (TYPEEXP INITIALIZATION ANSWER)                  (* (primary new %[ typeexp initialization optcatch %]) *)
          (PARSE.BIN 'NEW)
          (PARSE.BIN '%[)
          (SETQ TYPEEXP (PARSE.TYPEEXP))
          (SETQ INITIALIZATION (PARSE.INITIALIZATION))
          (PARSE.OPTCATCH)
          (PARSE.BIN '%])
          (SETQ ANSWER (LIST 'create LHS TYPEEXP INITIALIZATION))
          (RETURN ANSWER])
```

(PARSE.PRIMARY.LHS.CONS

```
  [LAMBDA (LHS)                                            (* kbr%: "25-Nov-85 12:48")
    (PROG (EXPLIST OPTCATCH ANSWER)                        (* CEDAR (primary cons %[ explist optcatch %]) *)
          (PARSE.BIN 'CONS)
          (PARSE.BIN '%[)
          (SETQ EXPLIST (PARSE.EXPLIST))
          (PARSE.OPTCATCH)
          (PARSE.BIN '%])
          [SETQ ANSWER `(CONS ,LHS ,@EXPLIST]
          (RETURN ANSWER])

(PARSE.PRIMARY.LHS.LIST
  [LAMBDA (LHS)                                                    (* kbr%: "25-Nov-85 12:48")
    (PROG (EXPLIST OPTCATCH ANSWER)                                (* CEDAR (primary listcons %[ explist %]) *)
          (PARSE.BIN 'LIST)
          (PARSE.BIN '%[)
          (SETQ EXPLIST (PARSE.EXPLIST))
          (PARSE.BIN '%])
          [SETQ ANSWER `(LIST ,LHS ,@EXPLIST]
          (RETURN ANSWER])

(PARSE.QUALIFIER
  [LAMBDA NIL                                                      (* kbr%: "25-Nov-85 12:48")
    (PROG (ANSWER)                                                 (* (qualifier %. prefixop) (qualifier %.
                                                                   typeop) (qualifier %. SIZE) (qualifier %[ explist optcatch %])
                                                                   (qualifier %. id) (qualifier ^) *)
          [COND
             [(EQ PARSE.ATOM '%.)
              (PARSE.BIN)
              (COND
                 ((FMEMB PARSE.ATOM PARSE.PREFIXOPS)
                  (SETQ ANSWER (PARSE.PREFIXOP)))
                 ([OR (FMEMB PARSE.ATOM PARSE.TYPEOPS)
                      (AND (EQ PARSE.LANGUAGE 'CEDAR)
                           (EQ PARSE.ATOM 'CODE]
                  (SETQ ANSWER (PARSE.TYPEOP)))
                 ((EQ PARSE.ATOM 'SIZE)
                  (SETQ ANSWER (PARSE.BIN)))
                 ((EQ PARSE.ATOM 'FREE)                            (* (free lhs %. FREE) *)
                  (SETQ ANSWER (PARSE.BIN)))
                 ((EQ PARSE.ATOM 'NEW)                             (* (new lhs %. NEW) *)
                  (SETQ ANSWER (PARSE.BIN)))
                 ([AND (EQ PARSE.LANGUAGE 'CEDAR)
                       (FMEMB PARSE.ATOM '(LIST CONS]
                  (SETQ ANSWER (PARSE.BIN)))
                 (T (SETQ ANSWER (PARSE.BIN 'ID]
             ((EQ PARSE.ATOM '%[)
              (PARSE.BIN)
              (SETQ ANSWER (PARSE.EXPLIST))
              (PARSE.OPTCATCH)
              (PARSE.BIN '%])))
             (T (SETQ ANSWER (PARSE.BIN '^]
          (RETURN ANSWER])

(PARSE.LHS
  [LAMBDA NIL                                                      (* kbr%: "25-Nov-85 12:48")
    (PROG (EXP1 EXP2 OPTTYPE ANSWER)                               (* (lhs id) (lhs char) (lhs NARROW %[ exp opttype optcatch %])
                                                                   (lhs LOOPHOLE %[ exp opttype %])
                                                                   (lhs APPLY %[ exp %, exp optcatch %])
                                                                   (lhs %( exp %)) (lhs lhs qualifier) *)
          [COND
             ((EQ PARSE.ATOM 'TRUE)
              (PARSE.BIN)
              (SETQ ANSWER T))
             ((EQ PARSE.ATOM 'FALSE)
              (PARSE.BIN))
             ((EQ PARSE.ATOM 'NARROW)
              (PARSE.BIN)
              (PARSE.BIN '%[)
              (SETQ EXP1 (PARSE.EXP))
              (SETQ OPTTYPE (PARSE.OPTTYPE))
              (PARSE.OPTCATCH)
              (PARSE.BIN '%])
              (SETQ ANSWER (BUILD.COERCE EXP1 OPTTYPE)))
             ((EQ PARSE.ATOM 'LOOPHOLE)
              (PARSE.BIN)
              (PARSE.BIN '%[)
              (SETQ EXP1 (PARSE.EXP))
              (SETQ OPTTYPE (PARSE.OPTTYPE))
              (PARSE.BIN '%])
              (SETQ ANSWER (BUILD.COERCE EXP1 OPTTYPE)))
             ((EQ PARSE.ATOM 'APPLY)
              (PARSE.BIN)
              (PARSE.BIN '%[)
              (SETQ EXP1 (PARSE.EXP))
              (PARSE.BIN '%,)
              (SETQ EXP2 (PARSE.EXP))
              (PARSE.OPTCATCH)
              (PARSE.BIN '%])
              (SETQ ANSWER (LIST 'APPLY EXP1 EXP2)))
             ((EQ PARSE.ATOM '%()
              (PARSE.BIN)
```

```
               (SETQ EXP1 (PARSE.EXP))
               (PARSE.BIN '%))
               (SETQ ANSWER EXP1))
              ((EQ PARSE.CLASS 'ID)
               (SETQ ANSWER (PARSE.BIN)))
              ((EQ PARSE.CLASS 'CHAR)
               (SETQ ANSWER (BUILD.CHARCODE (PARSE.BIN]
          [while (PARSE.QUALIFIER.HERE) do (SETQ ANSWER (BUILD.QUALIFY ANSWER (PARSE.QUALIFIER]
          (RETURN ANSWER])
```

```
(PARSE.QUALIFIER.HERE
  [LAMBDA NIL                                                      (* kbr%: "25-Nov-85 12:48")
    (AND (FMEMB PARSE.ATOM PARSE.QUALIFIER.FIRST)
         (NOT (AND (EQ PARSE.ATOM '%.)
                   (OR (FMEMB PARSE.ATOM2 '(FREE NEW))
                       (AND (EQ PARSE.LANGUAGE 'CEDAR)
                            (FMEMB PARSE.ATOM2 '(CONS LIST])
```

```
(PARSE.OPTCATCH
  [LAMBDA NIL                                                      (* kbr%: "25-Nov-85 12:48")
    (PROG (CATCHLIST ANSWER)                                       (* (optcatch ! catchlist) (optcatch) *)
          [COND
             ((EQ PARSE.ATOM '!)
              (PARSE.BIN)
              (SETQ ANSWER (PARSE.CATCHLIST]
          (RETURN ANSWER])
```

```
(PARSE.TRANSFEROP
  [LAMBDA NIL                                                      (* kbr%: "25-Nov-85 12:48")
                                                                   (* (transferop SIGNAL) (transferop ERROR)
                                                                   (transferop START) (transferop JOIN)
                                                                   (transferop NEW) (transferop FORK) *)
    (PARSE.BIN])
```

```
(PARSE.PREFIXOP
  [LAMBDA NIL                                                      (* kbr%: "25-Nov-85 12:48")
                                                                   (* (prefixop LONG) (prefixop ABS)
                                                                   (prefixop PRED) (prefixop SUCC)
                                                                   (prefixop ORD) (prefixop MIN) (prefixop MAX)
                                                                   (prefixop BASE) (prefixop LENGTH) *)
    (PARSE.BIN])
```

```
(PARSE.TYPEOP
  [LAMBDA NIL                                                      (* kbr%: "25-Nov-85 12:48")
                                                                   (* BOTH (typeop FIRST) (typeop LAST)
                                                                   (typeop NIL) *)
                                                                   (* CEDAR (typeop CODE) *)
    (COND
       ((EQ PARSE.ATOM 'CODE)
        (PARSE.THISIS.CEDAR)))
    (PARSE.BIN])
```

```
(PARSE.DESCLIST
  [LAMBDA NIL                                                      (* kbr%: "25-Nov-85 12:48")
    (PROG (EXP1 EXP2 OPTTYPE ANSWER)                               (* (desclist exp %, exp opttype) (desclist exp) *)
          (SETQ EXP1 (PARSE.EXP))
          [COND
             ((EQ PARSE.ATOM '%,)
              (PARSE.BIN)
              (SETQ EXP2 (PARSE.EXP))
              (SETQ OPTTYPE (PARSE.OPTTYPE))
              (SETQ ANSWER (LIST 'desclist EXP1 EXP2 OPTTYPE))
              (RETURN ANSWER))
             (T (SETQ ANSWER (LIST 'desclist EXP1]
          (RETURN ANSWER])
```

```
(PARSE.DIRECTORY
  [LAMBDA NIL                                                      (* kbr%: "25-Nov-85 12:48")
    (PROG NIL                                                      (* (directory DIRECTORY ;) (directory DIRECTORY includelist ;)
                                                                   (directory) *)
          (COND
             ((EQ PARSE.ATOM 'DIRECTORY)
              (PARSE.BIN)
              (COND
                 ((EQ PARSE.ATOM ';))
                 (T (PARSE.INCLUDELIST)
                    (PARSE.BIN ';])
```

```
(PARSE.IMPORTS
  [LAMBDA NIL                                                   (* kbr%: "25-Nov-85 12:48")
    (PROG NIL                                                   (* (imports IMPORTS) (imports IMPORTS modulelist)
                                                                (imports) *)
          (COND
             ((EQ PARSE.ATOM 'IMPORTS)
              (PARSE.BIN)
              (PARSE.MODULELIST])

(PARSE.POINTERPREFIX
  [LAMBDA NIL                                                   (* kbr%: "25-Nov-85 12:48")
    (PROG (ANSWER)                                              (* (pointerprefix POINTER) (pointerprefix POINTER interval) *)
          (PARSE.BIN 'POINTER)
          [COND
             ((FMEMB PARSE.ATOM PARSE.INTERVAL.FIRST)
              (SETQ ANSWER (LIST 'POINTER (PARSE.INTERVAL]
          (RETURN ANSWER])

(PARSE.EXPORTS
  [LAMBDA NIL                                                   (* kbr%: "25-Nov-85 12:48")
    (PROG (MODULELIST ANSWER)                                   (* (exports EXPORTS) (exports EXPORTS modulelist)
                                                                (exports) *)
          (COND
             ((EQ PARSE.ATOM 'EXPORTS)
              (PARSE.BIN)
              (BUILD.STORE.EXPORTS (PARSE.MODULELIST])

(PARSE.FIELDLIST
  [LAMBDA (KIND)                                                (* kbr%: "25-Nov-85 12:48")
    (PROG (ANSWER)                                              (* (fieldlist %[ %]) (fieldlist %[ pairlist %])
                                                                (fieldlist %[ typelist %]) *)
          (PARSE.BIN '%[)
          [COND
             ((NOT (EQ PARSE.ATOM '%]))
              (COND
                 [[AND (EQ PARSE.CLASS 'ID)
                       (NOT (FMEMB PARSE.ATOM PARSE.PREDEFINED.TYPES))
                       (FMEMB PARSE.ATOM2 '(%( %, %:]
                  (SETQ ANSWER (PARSE.PAIRLIST 'FIELDLIST]
                 (T (SETQ ANSWER (PARSE.TYPELIST]
          (PARSE.BIN '%])
          (RETURN ANSWER])

(PARSE.USING
  [LAMBDA NIL                                                   (* kbr%: "25-Nov-85 12:48")
    (PROG (IDLIST)                                              (* (using USING %[ %]) (using USING %[ idlist %])
                                                                (using) *)
          [COND
             ((EQ PARSE.ATOM 'USING)
              (PARSE.BIN)
              (PARSE.BIN '%[)
              (COND
                 ((EQ PARSE.ATOM '%])
                  (PARSE.BIN))
                 (T (SETQ IDLIST (PARSE.IDLIST))
                    (PARSE.BIN '%]]
          (RETURN IDLIST])

(PARSE.CATCHHEAD
  [LAMBDA NIL                                                   (* kbr%: "25-Nov-85 12:48")
    (PROG (CATCHCASES ANSWER)                                   (* (catchhead) (catchhead catchhead catchcase ;) *)
          (COND
             ((FMEMB PARSE.ATOM PARSE.CATCHLIST.FOLLOW)
              (RETURN)))
          (push CATCHCASES (PARSE.CATCHCASE))
          (while (EQ PARSE.ATOM '; ) do (PARSE.BIN)
                                        (COND
                                           ((FMEMB PARSE.ATOM PARSE.CATCHLIST.FOLLOW)
                                            (RETURN)))
                                        (push CATCHCASES (PARSE.CATCHCASE)))
          (SETQ ANSWER (DREVERSE CATCHCASES))
          (RETURN ANSWER])

(PARSE.DECLIST
  [LAMBDA NIL                                                   (* kbr%: "25-Nov-85 12:48")
    (PROG (VARLIST)                                             (* (declist declaration) (declist declist ;
                                                                declaration) *)
          (SETQ VARLIST (PARSE.DECLARATION))
          [do (COND
                 ((EQ PARSE.ATOM '; )
                  (PARSE.BIN))
                 ((FMEMB PARSE.ATOM PARSE.DECLIST.FOLLOW)
                  (RETURN))
                (T (SHOULDNT "PARSE.DECLIST")))
             (COND
                ([NOT (AND (EQ PARSE.CLASS 'ID)
                           (FMEMB PARSE.ATOM2 '(%, %:]
                  (RETURN)))
             (SETQ VARLIST (NCONC VARLIST (PARSE.DECLARATION]
        (BUILD.STORE.VARLIST VARLIST])

(PARSE.PAIRLIST
  [LAMBDA (KIND)                                         (* kbr%: "25-Nov-85 12:48")
    (PROG (PAIRITEMS ANSWER)                             (* (pairlist pairitem) (pairlist pairlist %, pairitem) *)
                                                         (* PARSE.PAIRITEM returns a list of PAIRITEM records.
                                                         *)
          (SETQ PAIRITEMS (PARSE.PAIRITEM KIND))
          (COND
             ((type? TYPELIST PAIRITEMS)                 (* Thought we we're parsing a pairlist, but found a typelist.
                                                         *)
              (RETURN PAIRITEMS)))
          [while (EQ PARSE.ATOM '%,) do (PARSE.BIN)
                                        (SETQ PAIRITEMS (NCONC PAIRITEMS (PARSE.PAIRITEM KIND]
          (SETQ ANSWER (create PAIRLIST
                               ITEMS _ PAIRITEMS))
          (RETURN ANSWER])

(PARSE.VARIANTLIST
  [LAMBDA NIL                                            (* kbr%: "25-Nov-85 12:48")
    (PROG (VARIANTITEMS ANSWER)                          (* (variantlist variantitem) (variantlist variantlist %, variantitem) *)
          (push VARIANTITEMS (PARSE.VARIANTITEM))
          (while (EQ PARSE.ATOM '%,) do (PARSE.BIN)
                                        (push VARIANTITEMS (PARSE.VARIANTITEM)))
          (SETQ ANSWER (CONS 'variantlist (DREVERSE VARIANTITEMS)))
          (RETURN ANSWER])

(PARSE.ORDERLIST
  [LAMBDA NIL                                            (* kbr%: "25-Nov-85 12:48")
    (PROG (OPTEXPS ANSWER)                               (* (orderlist optexp) (orderlist orderlist %, optexp) *)
          (COND
             ((FMEMB PARSE.ATOM PARSE.ORDERLIST.FOLLOW)
              (RETURN)))
          (push OPTEXPS (PARSE.OPTEXP))
          (while (EQ PARSE.ATOM '%,) do (PARSE.BIN)
                                        (push OPTEXPS (PARSE.OPTEXP)))
          (SETQ ANSWER (create ORDERLIST
                               ITEMS _ (DREVERSE OPTEXPS)))
          (RETURN ANSWER])

(PARSE.LHSLIST
  [LAMBDA NIL                                            (* kbr%: "25-Nov-85 12:48")
    (PROG (LHSS ANSWER)                                  (* (lhslist lhs) (lhslist lhslist %, lhs) *)
          (push LHSS (PARSE.LHS))
          (while (EQ PARSE.ATOM '%,) do (PARSE.BIN)
                                        (push LHSS (PARSE.LHS)))
          (SETQ ANSWER (DREVERSE LHSS))
          (RETURN ANSWER])

(PARSE.INCLUDELIST
  [LAMBDA NIL                                            (* kbr%: "25-Nov-85 12:48")
    (PROG NIL                                            (* (includelist includeitem) (includelist includelist %, includeitem)
     *)
          (PARSE.INCLUDEITEM)
          (while (EQ PARSE.ATOM '%,) do (PARSE.BIN)
                                        (PARSE.INCLUDEITEM])

(PARSE.MODULELIST
  [LAMBDA NIL                                            (* kbr%: "25-Nov-85 12:48")
    (PROG (MODULEITEMS ANSWER)                           (* (modulelist moduleitem) (modulelist modulelist %, moduleitem)
     *)
          (COND
             ((FMEMB PARSE.ATOM PARSE.MODULELIST.FOLLOW)
              (RETURN NIL)))
          (push MODULEITEMS (PARSE.MODULEITEM))
          (while (EQ PARSE.ATOM '%,) do (PARSE.BIN)
                                        (push MODULEITEMS (PARSE.MODULEITEM)))
          (SETQ ANSWER (DREVERSE MODULEITEMS))
          (RETURN ANSWER])

(PARSE.ELEMENTLIST
  [LAMBDA NIL                                            (* kbr%: "25-Nov-85 12:48")
    (PROG (ELEMENTS ANSWER)                              (* (elementlist element) (elementlist elementlist %, element) *)
          (push ELEMENTS (PARSE.ELEMENT))
          (while (EQ PARSE.ATOM '%,) do (PARSE.BIN)
                                        (push ELEMENTS (PARSE.ELEMENT)))
          (SETQ ANSWER (DREVERSE ELEMENTS))
          (RETURN ANSWER])

(PARSE.BINDLIST
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 12:48")
    (PROG (BINDITEMS ANSWER)                                         (* (bindlist binditem) (bindlist bindlist %, binditem) *)
          (push BINDITEMS (PARSE.BINDITEM))
          (while (EQ PARSE.ATOM '%,) do (PARSE.BIN)
                                        (push BINDITEMS (PARSE.BINDITEM)))
          (SETQ ANSWER (DREVERSE BINDITEMS))
          (RETURN ANSWER])

(PARSE.STATEMENTLIST
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 12:48")
    (PROG (STATEMENTS ANSWER)                                        (* (statementlist statement) (statementlist statementlist ;
                                                                     statement) *)
          (COND
             ((FMEMB PARSE.ATOM PARSE.STATEMENTLIST.FOLLOW)
              (RETURN)))
          (push STATEMENTS (PARSE.STATEMENT))
          (do (COND
                 ((EQ PARSE.ATOM ';)
                  (PARSE.BIN)))
              (COND
                 ((FMEMB PARSE.ATOM PARSE.STATEMENTLIST.FOLLOW)
                  (RETURN)))
              (push STATEMENTS (PARSE.STATEMENT)))
          (SETQ ANSWER (DREVERSE STATEMENTS))
          (RETURN ANSWER])

(PARSE.CASESTMTLIST
  [LAMBDA (CASEHEAD)                                                 (* kbr%: "25-Nov-85 12:48")
    (PROG (CASESTMTITEMS ANSWER)                                     (* (casestmtlist casestmtitem) (casestmtlist casestmtlist ;
                                                                     casestmtitem) *)
          (push CASESTMTITEMS (PARSE.CASESTMTITEM CASEHEAD))
          (do (COND
                 ((EQ PARSE.ATOM ';)
                  (PARSE.BIN)))
              (COND
                 ((EQ PARSE.ATOM 'ENDCASE)
                  (RETURN)))
              (push CASESTMTITEMS (PARSE.CASESTMTITEM CASEHEAD)))
          (SETQ ANSWER (DREVERSE CASESTMTITEMS))
          (RETURN ANSWER])

(PARSE.CASELABEL
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 12:48")
    (PROG (CASETESTS IDENT TYPEEXP ANSWER)                           (* (caselabel ident typeexp) (caselabel caselabel')
                                                                     (caselabel' casetest) (caselabel' caselabel' %, casetest) *)
          (COND
             ([AND (EQ PARSE.CLASS 'ID)
                   (FMEMB PARSE.ATOM2 '(%: %(]
              (SETQ IDENT (PARSE.IDENT))
              (SETQ TYPEEXP (PARSE.TYPEEXP))
              (SETQ ANSWER (LIST (BUILD.ISTYPE IDENT TYPEEXP)))
              (BUILD.INITIALIZE.VAR IDENT TYPEEXP NIL BUILD.CURRENT.SCOPE)
              (RETURN ANSWER)))
          (push CASETESTS (PARSE.CASETEST))
          (while (EQ PARSE.ATOM '%,) do (PARSE.BIN)
                                        (push CASETESTS (PARSE.CASETEST)))
          (SETQ ANSWER (DREVERSE CASETESTS))
          (RETURN ANSWER])

(PARSE.EXITLIST
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 12:48")
    (PROG (EXITITEMS ANSWER)                                         (* (exitlist exititem) (exitlist exitlist ;
                                                                     exititem) *)
          (COND
             ((FMEMB PARSE.ATOM PARSE.EXITLIST.FOLLOW)
              (RETURN)))
          (push EXITITEMS (PARSE.EXITITEM))
          (do (COND
                 ((EQ PARSE.ATOM ';)
                  (PARSE.BIN)))
              (COND
                 ((FMEMB PARSE.ATOM PARSE.EXITLIST.FOLLOW)
                  (RETURN)))
              (push EXITITEMS (PARSE.EXITITEM)))
          (SETQ ANSWER (DREVERSE EXITITEMS))
          (RETURN ANSWER])

(PARSE.KEYLIST
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
    (PROG (KEYITEMS ANSWER)                                (* (keylist keyitem) (keylist keylist %, keyitem) *)
          (push KEYITEMS (PARSE.KEYITEM))
          (while (EQ PARSE.ATOM '%,) do (PARSE.BIN)
                                        (push KEYITEMS (PARSE.KEYITEM)))
          (SETQ ANSWER (create KEYLIST
                              ITEMS _ (DREVERSE KEYITEMS)))
          (RETURN ANSWER])

(PARSE.CASEEXPLIST
  [LAMBDA (CASEHEAD)                                       (* kbr%: "25-Nov-85 12:48")
    (PROG (CASEEXPITEMS ANSWER)                            (* (caseexplist caseexpitem) (caseexplist caseexplist %,
                                                           caseexpitem) *)
          (push CASEEXPITEMS (PARSE.CASEEXPITEM CASEHEAD))
          (do (COND
                 ((EQ PARSE.ATOM '%,)
                  (PARSE.BIN)))
              (COND
                 ((EQ PARSE.ATOM 'ENDCASE)
                  (RETURN)))
              (push CASEEXPITEMS (PARSE.CASEEXPITEM CASEHEAD)))
          (SETQ ANSWER (DREVERSE CASEEXPITEMS))
          (RETURN ANSWER])

(PARSE.EXPLIST
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
                                                           (* (explist orderlist) (explist keylist) *)
    (PROG (ORDERLIST KEYLIST ANSWER)
          [COND
             ((AND (EQ PARSE.CLASS 'ID)
                   (EQ PARSE.ATOM2 '%:))
              (SETQ ANSWER (PARSE.KEYLIST)))
             (T (SETQ ANSWER (PARSE.ORDERLIST]
          (RETURN ANSWER])

(PARSE.OPEN
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
    (PROG (BINDLIST)                                       (* (open OPEN bindlist ;) (open) *)
          [COND
             ((EQ PARSE.ATOM 'OPEN)
              (PARSE.BIN)
              (SETQ BINDLIST (PARSE.BINDLIST))
              (PARSE.BIN ';]
          (RETURN BINDLIST])

(PARSE.CLASS
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
                                                           (* (class PROGRAM) (class MONITOR) *)
    (PARSE.BIN '(MONITOR PROGRAM])

(PARSE.CASEHEAD
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
    (PROG (ID EXP OPTEXP BINDITEM OPTEXP ANSWER)           (* (casehead SELECT exp FROM)
                                                           (casehead WITH binditem SELECT optexp FROM) *)
          [COND
             ((EQ PARSE.ATOM 'SELECT)
              (PARSE.BIN)
              (SETQ EXP (PARSE.EXP))
              (PARSE.BIN 'FROM))
             (T (PARSE.BIN 'WITH)
                (SETQ BINDITEM (PARSE.BINDITEM))
                (SETQ ID (fetch (BINDITEM ID) of BINDITEM))
                (SETQ EXP (fetch (BINDITEM EXP) of BINDITEM))
                (PARSE.BIN 'SELECT)
                (SETQ OPTEXP (PARSE.OPTEXP))
                (PARSE.BIN 'FROM]
          (SETQ ANSWER (create CASEHEAD
                              ID _ ID
                              EXP _ EXP
                              OPTEXP _ OPTEXP))
          (RETURN ANSWER])

(PARSE.READONLY
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
                                                           (* (readonly READONLY) (readonly) *)
    (COND
       ((EQ PARSE.ATOM 'READONLY)
        (PARSE.BIN])

(PARSE.ORDERED
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
                                                           (* (ordered ORDERED) (ordered) *)
    (COND
       ((EQ PARSE.ATOM 'ORDERED)
        (PARSE.BIN])

(PARSE.BASE
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
                                                           (* (base BASE) (base) *)
    (COND
       ((EQ PARSE.ATOM 'BASE)
        (PARSE.BIN])

(PARSE.PACKED
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
                                                           (* (packed PACKED) (packed) *)
    (COND
       ((EQ PARSE.ATOM 'PACKED)
        (PARSE.BIN])

(PARSE.HEAP
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
                                                           (* BOTH (heap UNCOUNTED) *)
                                                           (* CEDAR (heap) *)
    (COND
       ((EQ PARSE.ATOM 'UNCOUNTED)
        (PARSE.BIN))
       (T (PARSE.THISIS.CEDAR])

(PARSE.INLINE
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
                                                           (* (inline INLINE) (inline) *)
    (COND
       ((EQ PARSE.ATOM 'INLINE)
        (PARSE.BIN])

(PARSE.ARGUMENTS
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
    (PROG NIL                                              (* (arguments arglist returnlist) *)
          (PARSE.ARGLIST)
          (PARSE.RETURNLIST])

(PARSE.INTERFACE
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
    (PROG NIL                                              (* (interface imports exports shares) *)
          (PARSE.IMPORTS)
          (PARSE.EXPORTS)
          (PARSE.SHARES])

(PARSE.SHARES
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
                                                           (* (shares SHARES idlist) (shares) *)
    (COND
       ((EQ PARSE.ATOM 'SHARES)
        (CONS (PARSE.BIN)
              (PARSE.IDLIST])

(PARSE.DEFAULT
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
                                                           (* (default _ defaultopt) (default) *)
    (COND
       ((EQ PARSE.ATOM '_)
        (PARSE.BIN)
        (PARSE.DEFAULTOPT])

(PARSE.OPTSIZE
  [LAMBDA NIL                                              (* kbr%: "25-Nov-85 12:48")
    (PROG (EXP ANSWER)                                     (* (optsize %[ exp %]) (optsize) *)
          (COND
             ((EQ PARSE.ATOM '%[)
              (PARSE.BIN)
              (SETQ EXP (PARSE.EXP))
              (PARSE.BIN '%])
              (SETQ ANSWER EXP)))
          (RETURN ANSWER])

(PARSE.BOUNDS
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:48")
    (PROG (EXP1 EXP2 ANSWER)                                     (* (bounds exp |..| exp) *)
          (SETQ EXP1 (PARSE.EXP))
          (PARSE.BIN '|..|)
          (SETQ EXP2 (PARSE.EXP))
          (SETQ ANSWER (LIST EXP1 EXP2))
          (RETURN ANSWER])

(PARSE.LENGTH
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:48")
    (PROG (EXP ANSWER)                                           (* (length %[ exp %]) *)
          (PARSE.BIN '%[)
          (SETQ EXP (PARSE.EXP))
          (PARSE.BIN '%])
          (SETQ ANSWER EXP)
          (RETURN ANSWER])

(PARSE.INDEXTYPE
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:48")
                                                                 (* (indextype typeexp) (indextype) *)
    (COND
       ((NOT (EQ PARSE.ATOM 'OF))
        (PARSE.TYPEEXP])

(PARSE.ELSEPART
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:48")
                                                                 (* (elsepart ELSE statement) (elsepart) *)
    (COND
       ((EQ PARSE.ATOM 'ELSE)
        (PARSE.BIN)
        (PARSE.STATEMENT])

(PARSE.OTHERPART
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:48")
                                                                 (* (otherpart => statement) (otherpart) *)
    (COND
       ((EQ PARSE.ATOM '=>)
        (PARSE.BIN)
        (PARSE.STATEMENT])

(PARSE.FREE
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:48")
    (PROG (LHS ANSWER)                                           (* (free lhs %. FREE) *)
          (SETQ LHS (PARSE.LHS))
          (PARSE.BIN '%.)
          (PARSE.BIN 'FREE)
          (SETQ ANSWER (LIST 'FREE LHS))
          (RETURN ANSWER])

(PARSE.CATCHANY
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:48")
    (PROG (STATEMENT ANSWER)                                     (* (catchany ANY => statement) *)
          (PARSE.BIN 'ANY)
          (PARSE.BIN '=>)
          (SETQ STATEMENT (PARSE.STATEMENT))
          (SETQ ANSWER (LIST 'ANY STATEMENT))
          (RETURN ANSWER])

(PARSE.NOT
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:48")
                                                                 (* (not ~) (not NOT) *)
    (PARSE.BIN])

(PARSE.NEW
  [LAMBDA NIL                                                    (* kbr%: "25-Nov-85 12:48")
    (PROG NIL                                                    (* BOTH (new lhs %. NEW) *)
                                                                 (* CEDAR (new NEW) *)
          (COND
             ((AND (EQ PARSE.ATOM 'NEW)
                   (EQ PARSE.LANGUAGE 'CEDAR))
              (PARSE.BIN))
             (T                                                  (* Throw away lhs. Interlisp doesn't have separate storage
                                                                 "zone" (QUOTE s.) *)
                (PARSE.LHS)
                (PARSE.BIN '%.)
                (PARSE.BIN 'NEW])

(PARSE.OPTTYPE
  [LAMBDA NIL                                                      (* kbr%: "25-Nov-85 12:48")
                                                                   (* (opttype %, typeexp) (opttype) *)
    (COND
       ((EQ PARSE.ATOM '%,)
        (PARSE.BIN)
        (PARSE.TYPEEXP))
       (T 'ANY])

(PARSE.ARGLIST
  [LAMBDA NIL                                                      (* kbr%: "25-Nov-85 12:48")
                                                                   (* BOTH (arglist fieldlist) (arglist) *)
                                                                   (* CEDAR (arglist ANY) *)

    (PROG (ARGLIST)
          [SETQ ARGLIST (COND
                           ((EQ PARSE.ATOM '%[)
                            (COND
                               ((EQ PARSE.ATOM 'ANY)
                                (PARSE.THISIS.CEDAR)
                                (PARSE.BIN))
                               (T (PARSE.FIELDLIST 'ARGLIST]
          (BUILD.STORE.ARGLIST ARGLIST])

(PARSE.RETURNLIST
  [LAMBDA NIL                                                      (* kbr%: "25-Nov-85 12:48")
                                                                   (* BOTH (returnlist RETURNS fieldlist)
                                                                   (returnlist) *)
                                                                   (* CEDAR (returnlist RETURNS ANY) *)
    (PROG (RETURNLIST)
          [SETQ RETURNLIST (COND
                              ((EQ PARSE.ATOM 'RETURNS)
                               (PARSE.BIN)
                               (COND
                                  ((EQ PARSE.ATOM 'ANY)
                                   (PARSE.THISIS.CEDAR)
                                   (PARSE.BIN))
                                  (T (PARSE.FIELDLIST 'RETURNLIST]
          (BUILD.STORE.RETURNLIST RETURNLIST])
)

;; BUILD

(RPAQ? BUILD.NEXT.SCOPE NIL)

(RPAQ? BUILD.CURRENT.SCOPE NIL)

(RPAQ? BUILD.SCOPE.STACK NIL)

(RPAQ? BUILD.PREFIX NIL)

(RPAQ? BUILD.FILECOMS NIL)

(RPAQ? BUILD.BOOLEAN.FNS '(AND OR NOT type? IGREATERP ILESSP IGEQ ILEQ IEQP ZEROP MINUSP EVENP ODDP FGREATERP
                             FLESSP FEQP GREATERP LESSP GEQ LEQ))

(RPAQ? BUILD.CARDINAL.FNS '(ADD1 CHARCODE FIX GCD IDIFFERENCE IMAX IMIN IMINUS IMOD IPLUS IQUOTIENT IREMAINDER
                              ITIMES LOGAND LOGNOT LOGOR LOGXOR NTHCHARCODE SUB1))

(RPAQ? BUILD.MIXED.FNS '(ABS DIFFERENCE EXPT MAX MIN MINUS MOD PLUS QUOTIENT REMAINDER TIMES))

(RPAQ? BUILD.REAL.FNS '(ANTILOG ARCCOS ARCSIN ARCTAN ARCTAN2 COS FDIFFERENCE FLOAT FMAX FMIN FMINUS FMOD FPLUS
                          FQUOTIENT FREMAINDER FTIMES LOG SIN SQRT TAN))

(RPAQ? BUILD.QUALIFY.WORDS '(FREE NEW SIZE))

(RPAQ? BUILD.CARDINAL.ARITHOP.ALIST
       (LIST (CONS '= 'IEQP)
             (CONS '%# 'IEQP)
             (CONS '< 'ILESSP)
             (CONS '<= 'ILEQ)
             (CONS '> 'IGREATERP)
             (CONS '>= 'IGEQ)
             (CONS '+ 'IPLUS)
             (CONS '- 'IDIFFERENCE)
             (CONS '* 'ITIMES)
             (CONS '/ 'IQUOTIENT)
             (CONS '0- 'IMINUS)
             (CONS 'MAX 'IMAX)
             (CONS 'MIN 'IMIN)
             (CONS 'MOD 'IMOD)))
```

```
(RPAQ? BUILD.MIXED.ARITHOP.ALIST
       (LIST (CONS '= 'EQP)
             (CONS '%# 'EQP)
             (CONS '< 'LESSP)
             (CONS '<= 'GREATERP)
             (CONS '> 'GREATERP)
             (CONS '>= 'LESSP)
             (CONS '+ 'PLUS)
             (CONS '- 'DIFFERENCE)
             (CONS '* 'TIMES)
             (CONS '/ 'QUOTIENT)
             (CONS '0- 'MINUS)
             (CONS 'MAX 'MAX)
             (CONS 'MIN 'MIN)
             (CONS 'MOD 'IMOD)))

(RPAQ? BUILD.REAL.ARITHOP.ALIST
       (LIST (CONS '= 'FEQP)
             (CONS '%# 'FEQP)
             (CONS '< 'FLESSP)
             (CONS '<= 'FGREATERP)
             (CONS '> 'FGREATERP)
             (CONS '>= 'FLESSP)
             (CONS '+ 'FPLUS)
             (CONS '- 'FDIFFERENCE)
             (CONS '* 'FTIMES)
             (CONS '/ 'FQUOTIENT)
             (CONS '0- 'FMINUS)
             (CONS 'MAX 'FMAX)
             (CONS 'MIN 'FMIN)
             (CONS 'MOD 'IMOD)))

(RPAQ? BUILD.CARDINAL.TYPES '(CARDINAL CHAR CHARACTER INT INTEGER NAT WORD))

(DECLARE%: EVAL@COMPILE

[RECORD SCOPE (ID SYMBOLTABLE INITLIST ARGLIST VARLIST RETURNLIST RETURNS OPEN)
       (ACCESSFNS ((RETURNVARS (FOR PAIRITEM IN (fetch (PAIRLIST ITEMS) of (fetch (SCOPE RETURNLIST) of DATUM))
                                  collect (BUILD.LOCALVARID NIL (fetch (PAIRITEM ID) of PAIRITEM]
)

(DEFINEQ

(BUILD.INIT
  [LAMBDA (PREFIX)                                                   (* kbr%: "25-Nov-85 17:27")
    (PROG NIL
          (SETQ BUILD.PREFIX PREFIX)
          (SETQ BUILD.FILECOMS (FILECOMS (U-CASE PREFIX)))
          (SETTOPVAL BUILD.FILECOMS NIL)
          (printout T "Creating " BUILD.FILECOMS T)
          (SETQ BUILD.NEXT.SCOPE (create SCOPE
                                        ID _ 'MODULE))
          (SETQ BUILD.CURRENT.SCOPE NIL)
          (SETQ BUILD.SCOPE.STACK NIL])

(BUILD.PUSH.SCOPE
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 17:27")
    (PROG NIL
          (COND
             (BUILD.CURRENT.SCOPE (push BUILD.SCOPE.STACK BUILD.CURRENT.SCOPE)))
          (SETQ BUILD.CURRENT.SCOPE BUILD.NEXT.SCOPE)
          (SETQ BUILD.NEXT.SCOPE (create SCOPE))
          (RETURN (CAR BUILD.SCOPE.STACK])

(BUILD.POP.SCOPE
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 17:27")
    (PROG NIL
          (SETQ BUILD.NEXT.SCOPE BUILD.CURRENT.SCOPE)
          (SETQ BUILD.CURRENT.SCOPE (pop BUILD.SCOPE.STACK))
          (RETURN BUILD.CURRENT.SCOPE])

(BUILD.GC.SCOPE
  [LAMBDA NIL                                                        (* kbr%: "25-Nov-85 17:27")
    (PROG NIL
          (SETQ BUILD.NEXT.SCOPE (create SCOPE])

(BUILD.STORE.EXPORTS
  [LAMBDA (EXPORTS)                                                  (* kbr%: "25-Nov-85 17:27")
    (PROG NIL
          (COND
             (EXPORTS (SETQ BUILD.PREFIX (CAR EXPORTS])
(BUILD.STORE.IDENTLIST
  [LAMBDA (IDENTLIST)                                      (* kbr%: "25-Nov-85 17:27")
    (PROG NIL
          (replace (SCOPE ID) of BUILD.NEXT.SCOPE with (CAR IDENTLIST])

(BUILD.STORE.INTERFACES
  [LAMBDA (INTERFACES)                                     (* kbr%: "25-Nov-85 17:27")
    (PROG NIL
          (for INTERFACE in INTERFACES do (BUILD.STORE.INTERFACE INTERFACE])

(BUILD.STORE.INTERFACE
  [LAMBDA (INTERFACE)                                      (* kbr%: "25-Nov-85 17:27")
    (PROG NIL
          (PUTPROP INTERFACE 'MESA.INTERFACE T])

(BUILD.STORE.OPEN
  [LAMBDA (OPEN)                                           (* kbr%: "25-Nov-85 17:27")
    (PROG NIL
          (replace (SCOPE OPEN) of BUILD.NEXT.SCOPE with OPEN])

(BUILD.STORE.USING
  [LAMBDA (INTERFACE USING)                                (* kbr%: "25-Nov-85 17:27")
    (PROG NIL
          (for USE in USING do (PUTPROP USE 'MESA.USEDBY INTERFACE])

(BUILD.INITIALIZATION
  [LAMBDA (IDENTLIST TYPEEXP INITIALIZATION)               (* kbr%: "25-Nov-85 17:27")
    (PROG (ANSWER)
          (SELECTQ (BUILD.TYPEATOM TYPEEXP)
              (PROC (BUILD.INITIALIZE.FN (CAR IDENTLIST)
                           TYPEEXP INITIALIZATION))
              (MRECORD (BUILD.INITIALIZE.RECORD (CAR IDENTLIST)
                              TYPEEXP INITIALIZATION))
              (SETQ ANSWER (BUILD.INITIALIZE.VARS IDENTLIST TYPEEXP INITIALIZATION BUILD.CURRENT.SCOPE)))
          (RETURN ANSWER])

(BUILD.INITIALIZE.VARS
  [LAMBDA (IDENTLIST TYPEEXP INITIALIZATION SCOPE)         (* kbr%: "25-Nov-85 17:27")
    (PROG (ANSWER)
          (SETQ ANSWER (for ID in IDENTLIST collect (BUILD.INITIALIZE.VAR ID TYPEEXP INITIALIZATION SCOPE)))
          (RETURN ANSWER])

(BUILD.INITIALIZE.VAR
  [LAMBDA (ID TYPEEXP INITIALIZATION SCOPE)                (* kbr%: "25-Nov-85 17:27")
    (PROG (PAIRITEM)
          (SETQ PAIRITEM (create PAIRITEM
                                 ID _ ID
                                 TYPEEXP _ TYPEEXP
                                 DEFAULT _ INITIALIZATION))
          (replace (SCOPE SYMBOLTABLE) of SCOPE with (NCONC (fetch (SCOPE SYMBOLTABLE) of SCOPE)
                                                            (LIST PAIRITEM)))
          (COND
             ((NULL BUILD.SCOPE.STACK)
              (BUILD.ADD.TO.FILECOMS (LIST ID (BUILD.COERCE INITIALIZATION TYPEEXP))
                     'INITVARS)
              (PRIN1 ID T)
              (PRIN1 "," T)))
          (RETURN ID])

(BUILD.INITIALIZE.FN
  [LAMBDA (ID TYPEEXP INITIALIZATION)                      (* kbr%: "25-Nov-85 17:27")
    (PROG (PROCID ARGLIST RETURNLIST LAMBDA)
          (SETQ PROCID (BUILD.PROCID BUILD.PREFIX ID))
          (SETQ ARGLIST (fetch (SCOPE ARGLIST) of BUILD.NEXT.SCOPE))
          (SETQ RETURNLIST (fetch (SCOPE RETURNLIST) of BUILD.NEXT.SCOPE))
          (PUTPROP ID 'MESA.USEDBY BUILD.PREFIX)
          (PUTPROP PROCID 'MESA.FN T)
          (PUTPROP PROCID 'MESA.ARGLIST ARGLIST)
          (PUTPROP PROCID 'MESA.RETURNLIST RETURNLIST)
          (SETQ LAMBDA (BUILD.LAMBDA ARGLIST INITIALIZATION))
          (PUTD PROCID LAMBDA)
          (BUILD.ADD.TO.FILECOMS PROCID 'FNS)
          (BUILD.GC.SCOPE)
          (PRIN1 ID T)
          (PRIN1 "," T])

(BUILD.INITIALIZE.RECORD
  [LAMBDA (ID TYPEEXP INITIALIZATION)                      (* kbr%: "25-Nov-85 17:27")
```

```
    (PROG (RECORDID FIELDLIST RECORD)
          (SETQ RECORDID (BUILD.RECORDID BUILD.PREFIX ID))
          (replace (MRECORD RECORDID) of TYPEEXP with RECORDID)
          (SETQ RECORD (BUILD.RECORD RECORDID TYPEEXP))
          (EVAL RECORD)
          (BUILD.ADD.TO.FILECOMS RECORDID 'RECORDS)
          (PUTPROP ID 'MESA.USEDBY BUILD.PREFIX)
          (PUTPROP RECORDID 'MESA.TYPE TYPEEXP)
          (PRIN1 ID T)
          (PRIN1 "," T])
```

(**BUILD.RECORD**

```
  [LAMBDA (RECORDID TYPEEXP)                                  (* kbr%: "25-Nov-85 17:27")
    (PROG (FIELDLIST FIELDS DEFAULTS ANSWER)
          (SETQ FIELDLIST (fetch (MRECORD FIELDLIST) of TYPEEXP))
          (COND
             [(NULL FIELDLIST)

          (* I'm not really sure what an empty FIELDLIST is supposed to get you in MESA/CEDAR.
          *)

              (RETURN `(TYPERECORD ,RECORDID]
             [(type? PAIRLIST FIELDLIST)
              (for ITEM in (REVERSE (fetch (PAIRLIST ITEMS) of FIELDLIST))
                 do (push FIELDS (fetch (PAIRITEM ID) of ITEM))
                    (COND
                       ((fetch (PAIRITEM DEFAULT) of ITEM)
                        (SETQ DEFAULTS (NCONC DEFAULTS `(,(fetch (PAIRITEM ID) of ITEM)

                                                         _
                                                         ,(BUILD.COERCE (fetch (PAIRITEM DEFAULT) of ITEM)
                                                                 (fetch (PAIRITEM TYPEEXP) of ITEM]
             [(type? TYPELIST FIELDLIST)
              (for ITEM in (REVERSE (fetch (TYPELIST ITEMS) of FIELDLIST)) as I from 1
                 do (push FIELDS (PACK* 'FIELD I))
                    (COND
                       ((fetch (TYPEITEM DEFAULT) of ITEM)
                        (SETQ DEFAULTS (NCONC DEFAULTS `(,(PACK* 'FIELD I)

                                                         _
                                                         ,(BUILD.COERCE (fetch (TYPEITEM DEFAULT) of ITEM)
                                                                 (fetch (TYPEITEM TYPEEXP) of ITEM]
             (T (SHOULDNT)))
          [SETQ ANSWER `(RECORD ,RECORDID ,FIELDS ,@DEFAULTS]
          (RETURN ANSWER])
```

(**BUILD.TYPE**

```
  [LAMBDA (IDENTLIST TYPEEXP DEFAULT)                         (* kbr%: "25-Nov-85 17:27")
    (PROG (ID TYPEID)
          (SELECTQ (BUILD.TYPEATOM TYPEEXP)
               (MRECORD (BUILD.INITIALIZE.RECORD (CAR IDENTLIST)
                               TYPEEXP DEFAULT))
               (PROGN (SETQ TYPEID (BUILD.TYPEID BUILD.PREFIX (CAR IDENTLIST)))
                      (COND
                         ((NOT (EQ TYPEID TYPEEXP))
                          (PUTPROP (CAR IDENTLIST)
                                 'MESA.USEDBY BUILD.PREFIX)
                          (PUTPROP TYPEID 'MESA.TYPE TYPEEXP])
```

(**BUILD.STORE.ARGLIST**

```
  [LAMBDA (ARGLIST)                                           (* kbr%: "25-Nov-85 17:27")
                                                              (* ARGLIST = args for coming function scope.
                                                              *)

    (PROG NIL
          (replace (SCOPE ARGLIST) of BUILD.NEXT.SCOPE with ARGLIST)
          (COND
             ((type? PAIRLIST ARGLIST)
              (BUILD.STORE.PAIRLIST ARGLIST])
```

(**BUILD.STORE.RETURNLIST**

```
  [LAMBDA (RETURNLIST)                                        (* kbr%: "25-Nov-85 17:27")
                                                              (* RETURNLIST = args for coming function scope.
                                                              *)

    (PROG NIL
          (replace (SCOPE RETURNLIST) of BUILD.NEXT.SCOPE with RETURNLIST)
          (COND
             ((type? PAIRLIST RETURNLIST)
              (BUILD.STORE.PAIRLIST RETURNLIST])
```

(**BUILD.STORE.PAIRLIST**

```
  [LAMBDA (PAIRLIST)                                          (* kbr%: "25-Nov-85 17:27")
                                                              (* PAIRLIST = args or return vals for coming function scope.
                                                              *)

    (PROG NIL
          (for PAIRITEM in (fetch (PAIRLIST ITEMS) of PAIRLIST) collect (BUILD.STORE.PAIRITEM PAIRITEM
                                                                               BUILD.NEXT.SCOPE])

(BUILD.STORE.PAIRITEM
  [LAMBDA (PAIRITEM SCOPE)                                        (* kbr%: "25-Nov-85 17:27")
    (PROG NIL
          (replace (SCOPE SYMBOLTABLE) of SCOPE with (NCONC (fetch (SCOPE SYMBOLTABLE) of SCOPE)
                                                            (LIST PAIRITEM)))
          (RETURN (fetch (PAIRITEM ID) of PAIRITEM])

(BUILD.STORE.VARLIST
  [LAMBDA (VARLIST)                                               (* kbr%: "25-Nov-85 17:27")
    (PROG NIL
          (replace (SCOPE VARLIST) of BUILD.CURRENT.SCOPE with VARLIST])

(BUILD.ID
  [LAMBDA (INTERFACE ID)                                          (* kbr%: "25-Nov-85 17:27")
    (PROG (ANSWER)
          (COND
             ((STRPOS "." ID)
              (RETURN ID)))
          [SETQ INTERFACE (OR INTERFACE (GETPROP ID 'MESA.USEDBY]
          (SETQ ANSWER (COND
                          (INTERFACE (PACK* INTERFACE "." ID))
                          (T ID)))
          (RETURN ANSWER])

(BUILD.FIELDID
  [LAMBDA (INTERFACE ID)                                          (* kbr%: "25-Nov-85 17:27")
    ID])

(BUILD.PROCID
  [LAMBDA (INTERFACE ID)                                          (* kbr%: "25-Nov-85 17:27")
    (BUILD.ID INTERFACE ID])

(BUILD.RECORDID
  [LAMBDA (INTERFACE ID)                                          (* kbr%: "25-Nov-85 17:27")
    (BUILD.ID INTERFACE ID])

(BUILD.TYPEID
  [LAMBDA (INTERFACE ID)                                          (* kbr%: "25-Nov-85 17:27")
    (COND
       ((FMEMB ID PARSE.PREDEFINED.TYPES)
        ID)
       (T (BUILD.ID INTERFACE ID])

(BUILD.VARID
  [LAMBDA (INTERFACE ID)                                          (* kbr%: "25-Nov-85 17:27")
    (PROG (ANSWER)
          [SETQ ANSWER (COND
                          ((BUILD.LOOKUP ID)
                           (BUILD.LOCALVARID INTERFACE ID))
                          (T (BUILD.GLOBALVARID INTERFACE ID]
          (RETURN ANSWER])

(BUILD.LOCALVARID
  [LAMBDA (INTERFACE ID)                                          (* kbr%: "25-Nov-85 17:27")
    ID])

(BUILD.GLOBALVARID
  [LAMBDA (INTERFACE ID)                                          (* kbr%: "25-Nov-85 17:27")
    (BUILD.ID INTERFACE ID])

(BUILD.ULTIMATE.TYPE
  [LAMBDA (EXP)                                                   (* kbr%: "25-Nov-85 17:27")
    (BUILD.REFINE.TYPE (BUILD.IMMEDIATE.TYPE EXP])

(BUILD.REFINE.TYPE
  [LAMBDA (TYPE)                                                  (* kbr%: "25-Nov-85 17:27")
    (PROG (PAIRITEM NEXTTYPE)
      LOOP
          (SETQ NEXTTYPE (COND
                            ((OR (FMEMB TYPE '(ANY MPROC INTERFACE))
                                 (FMEMB TYPE PARSE.PREDEFINED.TYPES))
                             (SELECTQ TYPE
                                  (BOOL 'BOOLEAN)
                                  (CHAR 'CHARACTER)
                                  ((INT INTEGER NAT WORD)
                                       'CARDINAL)
                                  (StringBody 'STRING)
                                  (UNSPECIFIED 'ANY)
                                 TYPE))
                            ((LITATOM TYPE)
                             (OR (BUILD.LOOKUP.TYPE TYPE)
                                 (PROGN (printout T T TYPE " type unknown." T)
                                        (PUTPROP TYPE 'MESA.TYPE 'UNDECLARED)
                                        TYPE)))
                            ((type? MINTERVAL TYPE)
                             (fetch (MINTERVAL LBOUND) of TYPE))
                            ((type? MPOINTER TYPE)
                             (fetch (MPOINTER TYPE) of TYPE))
                            ((type? MREF TYPE)
                             (fetch (MREF TYPE) of TYPE))
                            (T TYPE)))
          (COND
             ((EQ NEXTTYPE 'UNDECLARED)
              (RETURN TYPE))
             ((NOT (EQ NEXTTYPE TYPE))
              (SETQ TYPE NEXTTYPE)
              (GO LOOP)))
          (RETURN TYPE])
```

(**BUILD.IMMEDIATE.TYPE**

```
  [LAMBDA (EXP)                                              (* kbr%: "25-Nov-85 17:27")
    (PROG (TYPE FN RECORDNAME FIELDNAME MRECORD FIELDLIST PAIRITEM)
          [SETQ TYPE (COND
                        ((OR (NULL EXP)
                             (EQ EXP T))
                         'BOOLEAN)
                        [(LITATOM EXP)
                         (OR (BUILD.LOOKUP.TYPE EXP)
                             (PROGN (printout T T EXP " type unknown." T)
                                    'ANY]
                        ((FIXP EXP)
                         'CARDINAL)
                        ((FLOATP EXP)
                         'REAL)
                        ((STRINGP EXP)
                         'STRING)
                        [(LISTP EXP)
                         (SETQ FN (CAR EXP))
                         (COND
                            ((EQ FN 'SETQ)
                             (BUILD.IMMEDIATE.TYPE (CADR EXP)))
                            [(EQ FN 'CAR)
                             (SETQ TYPE (BUILD.ULTIMATE.TYPE (CADR EXP)))
                             (COND
                                ((type? MLIST TYPE)
                                 (fetch (MLIST TYPE) of TYPE))
                                (T (printout T T EXP " type unknown." T)
                                   'ANY]
                            [(EQ FN 'CDR)
                             (SETQ TYPE (BUILD.ULTIMATE.TYPE (CADR EXP)))
                             (COND
                                ((type? MLIST TYPE)
                                 TYPE)
                                (T (printout T T EXP " type unknown." T)
                                   'ANY]
                            [(FMEMB FN '(CONS LIST))
                             (SETQ TYPE (BUILD.IMMEDIATE.TYPE (CADR EXP)))
                             (COND
                                (TYPE (create MLIST
                                             TYPE _ TYPE))
                                (T (printout T T EXP " type unknown." T)
                                   'ANY]
                            [(EQ FN 'COND)
                             (BUILD.IMMEDIATE.TYPE (CADR (CADR EXP]
                            [(EQ FN 'ELT)
                             (SETQ TYPE (BUILD.ULTIMATE.TYPE (CADR EXP)))
                             (COND
                                ((type? MARRAY TYPE)
                                 (fetch (MARRAY TYPE) of TYPE))
                                (T (printout T T EXP " type unknown." T)
                                   'ANY]
                            ((EQ FN 'create)
                             (CADR EXP))
                            [(EQ FN 'fetch)
                             (SETQ RECORDNAME (CAR (CADR EXP)))
                             (SETQ FIELDNAME (CADR (CADR EXP)))
                             (SETQ MRECORD (GETPROP RECORDNAME 'MESA.TYPE))
                             (COND
                                ((EQ MRECORD 'UNDECLARED)
                                 'ANY)
                                (T (SETQ FIELDLIST (fetch (MRECORD FIELDLIST) of MRECORD))
                                   (COND
                                      ((type? PAIRLIST FIELDLIST)
                                       (SETQ PAIRITEM (ASSOC FIELDNAME (fetch (PAIRLIST ITEMS) of FIELDLIST)))
                                       (fetch (PAIRITEM TYPEEXP) of PAIRITEM))
                                      (T (printout T T EXP " type unknown." T)
                                         'ANY]
                            ((FMEMB FN BUILD.BOOLEAN.FNS)
                             'BOOLEAN)
                            ((FMEMB FN BUILD.CARDINAL.FNS)
                             'CARDINAL)
                            ((FMEMB FN BUILD.MIXED.FNS)
                             'MIXED)
                            ((FMEMB FN BUILD.REAL.FNS)
                             'REAL)
                            (T (printout T T EXP " type unknown." T)
                               'ANY]
                        (T (printout T T EXP " type unknown." T)
                           'ANY]
          (RETURN TYPE])
```

(BUILD.LOOKUP.TYPE

```
  [LAMBDA (ID)                                                        (* kbr%: "25-Nov-85 17:27")
    (PROG (PAIRITEM TYPE)
          (SETQ PAIRITEM (BUILD.LOOKUP ID))
          (COND
             (PAIRITEM (SETQ TYPE (fetch (PAIRITEM TYPEEXP) of PAIRITEM))
                    (RETURN TYPE)))
          [SETQ TYPE (COND
                        ((GETPROP ID 'MESA.TYPE))
                        ((GETPROP ID 'MESA.USEDBY)
                         (BUILD.ID (GETPROP ID 'MESA.USEDBY)
                                ID))
                        ((GETPROP ID 'MESA.FN)
                         (RETURN 'MPROC))
                        ((GETPROP ID 'MESA.INTERFACE)
                         (RETURN 'INTERFACE]
          (RETURN TYPE])
```

(BUILD.LOOKUP

```
  [LAMBDA (ID)                                                        (* kbr%: "25-Nov-85 17:27")
    (PROG (ANSWER)
          [for SCOPE in (CONS BUILD.CURRENT.SCOPE BUILD.SCOPE.STACK) do (SETQ ANSWER (ASSOC ID
                                                                                          (fetch (SCOPE
                                                                                                  SYMBOLTABLE
                                                                                                   )
                                                                                             of SCOPE)))
                                                                                (COND
                                                                                   (ANSWER (RETURN]
          (RETURN ANSWER])
```

(BUILD.TYPEATOM

```
  [LAMBDA (TYPEEXP)                                                   (* kbr%: "25-Nov-85 17:27")
    (COND
       ((LITATOM TYPEEXP)
        TYPEEXP)
       (T (CAR TYPEEXP])
```

(BUILD.QUALIFY

```
  [LAMBDA (LHS QUALIFIER)                                             (* kbr%: "25-Nov-85 17:27")
    (PROG (TYPE TYPEATOM ANSWER)                                      (* (qualifier %. prefixop) (qualifier %.
                                                                      typeop) (qualifier %. SIZE) (qualifier %[ explist optcatch %])
                                                                      (qualifier %. id) (qualifier ^) *)
          [SETQ ANSWER (COND
                          ((FMEMB QUALIFIER PARSE.PREFIXOPS)
                           (BUILD.QUALIFY.PREFIXOP LHS QUALIFIER))
                          ((FMEMB QUALIFIER PARSE.TYPEOPS)
                           (BUILD.QUALIFY.TYPEOP LHS QUALIFIER))
                          ((EQ QUALIFIER 'SIZE)
                           (PACK* LHS "." QUALIFIER))
                          [(EQ QUALIFIER 'first)
                           `(CAR ,LHS]
                          [(EQ QUALIFIER 'rest)
                           `(CDR ,LHS]
                          ((OR (NULL QUALIFIER)
                               (LISTP QUALIFIER))
                           (BUILD.QUALIFY.EXPLIST LHS QUALIFIER))
                          ((EQ QUALIFIER '^)
                           LHS)
                          (T (BUILD.QUALIFY.ID LHS QUALIFIER]
          (RETURN ANSWER])
```

```
(BUILD.QUALIFY.PREFIXOP
  [LAMBDA (LHS QUALIFIER)                                     (* kbr%: "25-Nov-85 17:27")
    (SELECTQ QUALIFIER
        ((MAX MIN)
             (BUILD.ARITH.EXP* QUALIFIER LHS))
        (CONS QUALIFIER LHS])

(BUILD.QUALIFY.TYPEOP
  [LAMBDA (LHS QUALIFIER)                                     (* kbr%: "25-Nov-85 17:27")
    (CONS QUALIFIER LHS])

(BUILD.QUALIFY.EXPLIST
  [LAMBDA (LHS EXPLIST)                                       (* kbr%: "25-Nov-85 17:27")
                                                              (* Qualify LHS with EXPLIST qualifier.
                                                              *)

    (PROG (TYPE TYPEATOM EXPITEMS ANSWER)
          [COND
             ((LITATOM LHS)
              (SETQ LHS (BUILD.ID NIL LHS]
          (SETQ TYPE (BUILD.ULTIMATE.TYPE LHS))
          (SETQ TYPEATOM (BUILD.TYPEATOM TYPE))
          (SETQ EXPITEMS (fetch (EXPLIST ITEMS) of EXPLIST))
          [SETQ ANSWER (SELECTQ TYPEATOM
                            (MARRAY `(ELT ,LHS ,@EXPITEMS))
                            (MPROC (BUILD.CALL LHS EXPLIST))
                            (STRING `(NTHCHARCODE ,LHS ,@EXPITEMS))
                            (MRECORD                          (* Presumably record contains SEQUENCE.
                                                              *)
                                     `(ELT ,LHS ,@EXPITEMS))
                            (COND
                               ((AND (LISTP LHS)
                                     (IEQP (LENGTH LHS)
                                           2))                (* "ARG1.FN[ARG2,...,ARGn]" *)
                                (APPEND LHS EXPITEMS))
                               (T (printout T T LHS " qualified by " EXPLIST "?" T)
                                  (COND
                                     [(AND (type? ORDERLIST EXPLIST)
                                           (IEQP (LENGTH EXPITEMS)
                                                 1))          (* Guess array access. *)
                                      `(ELT ,LHS ,@EXPITEMS]
                                     (T (CONS LHS EXPITEMS]
          (RETURN ANSWER])

(BUILD.QUALIFY.ID
  [LAMBDA (LHS QUALIFIER)                                     (* kbr%: "25-Nov-85 17:27")
                                                              (* Qualify LHS with id QUALIFIER.
                                                              *)

    (PROG (TYPE TYPEATOM ANSWER)
          (SETQ TYPE (BUILD.ULTIMATE.TYPE LHS))
          (SETQ TYPEATOM (BUILD.TYPEATOM TYPE))
          [SETQ ANSWER (SELECTQ TYPEATOM
                            (MRECORD `(fetch (,(fetch (MRECORD RECORDID) of TYPE)
                                              ,QUALIFIER)
                                           of ,LHS))
                            (MARRAY (printout T T LHS " qualified by " QUALIFIER "?" T)
                                    `(ELT ,LHS ,QUALIFIER))
                            (INTERFACE (BUILD.ID LHS QUALIFIER))
                            (MPROC (COND
                                      (QUALIFIER (LIST LHS QUALIFIER))
                                      (T (LIST LHS))))
                            (STRING (printout T T LHS " qualified by " QUALIFIER "?" T)
                                    `(NTHCHARCODE ,LHS ,QUALIFIER))
                            (COND
                               [(EQ (GETPROP TYPE 'MESA.TYPE)
                                    'UNDECLARED)              (* Guess undeclared record. *)
                                `(fetch (,TYPE ,QUALIFIER) of ,LHS]
                               (T                             (* Guess undeclared fn. *)
                                  (LIST QUALIFIER LHS]
          (RETURN ANSWER])

(BUILD.ARITH.EXP1
  [LAMBDA (ARITHOP EXP1)                                      (* kbr%: "25-Nov-85 17:27")
    [COND
       ((EQ ARITHOP '-)
        (SETQ ARITHOP '0-]
    (BUILD.ARITH.EXP* ARITHOP (LIST EXP1])

(BUILD.ARITH.EXP2
  [LAMBDA (ARITHOP EXP1 EXP2)                                 (* kbr%: "25-Nov-85 17:27")
    (PROG (ANSWER)
          (SETQ ANSWER (BUILD.ARITH.EXP* ARITHOP (LIST EXP1 EXP2)))
          (RETURN ANSWER])
```

(BUILD.ARITH.EXP*

```
  [LAMBDA (ARITHOP EXPS)                                          (* kbr%: "25-Nov-85 17:27")
    (PROG (TYPE NEWARITHOP ANSWER)
          (SETQ TYPE (BUILD.STRONGEST.TYPE.AMONG EXPS))
          (SETQ NEWARITHOP (BUILD.COERCE.ARITHOP ARITHOP TYPE))
          [COND
             ((EQ TYPE 'REAL)
              (SETQ EXPS (for EXP in EXPS collect (COND
                                                     ((FIXP EXP)
                                                      (FLOAT EXP))
                                                     (T EXP]
          (SETQ ANSWER (CONS NEWARITHOP EXPS))
          [COND
             ((FMEMB NEWARITHOP '(IPLUS IDIFFERENCE))
              (SETQ ANSWER (BUILD.ARITH.ADD1SUB1 ANSWER)))
             [(AND (EQ ARITHOP '0-)
                   (NUMBERP (CADR ANSWER)))
              (SETQ ANSWER (APPLY* (CAR ANSWER)
                                  (CADR ANSWER]
             ([OR (EQ ARITHOP '%#)
                  (AND (FMEMB ARITHOP (LIST '<= '>=))
                       (NOT (EQ TYPE 'CARDINAL]
              (SETQ ANSWER (LIST 'NOT ANSWER]
          (RETURN ANSWER])
```

(BUILD.ARITH.ADD1SUB1

```
  [LAMBDA (EXP)                                                   (* kbr%: "25-Nov-85 17:27")
                                                                  (* Use ADD1 or SUB1 instead of IPLUS or IDIFFERENCE if
                                                                  possible. *)

    (PROG (FN EXP1 EXP2 ANSWER)
          (COND
             ((NOT (IEQP (FLENGTH EXP)
                         3))
              (RETURN EXP)))
          (SETQ FN (CAR EXP))
          (SETQ EXP1 (CADR EXP))
          (SETQ EXP2 (CADDR EXP))
          (COND
             [(EQ FN 'IPLUS)
              (COND
                 ((EQ EXP1 1)
                  (SETQ ANSWER (BUILD.ADD1 EXP2)))
                 ((EQ EXP2 1)
                  (SETQ ANSWER (BUILD.ADD1 EXP1)))
                 (T (SETQ ANSWER EXP]
             ((AND (EQ FN 'IDIFFERENCE)
                   (EQ EXP2 1))
              (SETQ ANSWER (BUILD.SUB1 EXP1)))
             (T (SETQ ANSWER EXP)))
          (RETURN ANSWER])
```

(BUILD.COERCE.ARITHOP

```
  [LAMBDA (ARITHOP TYPE)                                          (* kbr%: "25-Nov-85 17:27")
    (SELECTQ TYPE
        (CARDINAL (CDR (ASSOC ARITHOP BUILD.CARDINAL.ARITHOP.ALIST)))
        (MIXED (CDR (ASSOC ARITHOP BUILD.MIXED.ARITHOP.ALIST)))
        (REAL (CDR (ASSOC ARITHOP BUILD.REAL.ARITHOP.ALIST)))
        (SHOULDNT])
```

(BUILD.STRONGEST.TYPE.AMONG

```
  [LAMBDA (EXPS)                                                  (* kbr%: "25-Nov-85 17:27")
    (PROG (TYPE)
          (SETQ TYPE 'CARDINAL)
          [for EXP in EXPS while (NOT (EQ TYPE 'REAL)) do (SETQ TYPE (BUILD.STRONGEST.TYPE TYPE (
                                                                                        BUILD.ULTIMATE.TYPE
                                                                                          EXP]
          (RETURN TYPE])
```

(BUILD.STRONGEST.TYPE

```
  [LAMBDA (TYPE1 TYPE2)                                           (* kbr%: "25-Nov-85 17:27")
    [COND
       ((FMEMB TYPE1 BUILD.CARDINAL.TYPES)
        (SETQ TYPE1 'CARDINAL]
    [COND
       ((FMEMB TYPE2 BUILD.CARDINAL.TYPES)
        (SETQ TYPE2 'CARDINAL]
    (SELECTQ TYPE1
        (CARDINAL (SELECTQ TYPE2
                      (CARDINAL 'CARDINAL)
                      (REAL 'REAL)
                      'MIXED))
         (MIXED (SELECTQ TYPE2
                    (REAL 'REAL)
                    'MIXED))
         (REAL 'REAL)
         'MIXED])
```

(BUILD.COERCE

```
  [LAMBDA (EXP TYPE)                                                 (* kbr%: "25-Nov-85 17:27")
    (PROG (TYPEEXP ANSWER)
          (SETQ TYPEEXP (BUILD.REFINE.TYPE TYPE))
          (SETQ ANSWER (COND
                           ((type? MARRAY TYPEEXP)
                            (FRESHLINE T)
                            (printout T T "Coercion to " TYPE " array type." T)
                            (BUILD.COERCE.MARRAY EXP TYPEEXP))
                           ((type? MLIST TYPEEXP)
                            (BUILD.COERCE.MLIST EXP TYPEEXP))
                           ((type? EXPLIST EXP)
                            (BUILD.COERCE.EXPLIST EXP TYPEEXP))
                           (T EXP)))
          (RETURN ANSWER])
```

(BUILD.COERCE.MARRAY

```
  [LAMBDA (EXP MARRAY)                                               (* kbr%: "25-Nov-85 17:27")
    (PROG (TYPE ANSWER)                                              (* This is legal MESA/CEDAR code with no very elegant
                                                                     Interlisp translation. *)
          (SETQ TYPE (fetch (MARRAY TYPE) of MARRAY))
          (SETQ ANSWER (COND
                           [(type? EXPLIST EXP)                      (* Should be an ORDERLIST. *)
                            (CONS 'LIST (for ITEM in (fetch (EXPLIST ITEMS) of EXP) collect (BUILD.COERCE ITEM TYPE]
                           (T                                        (* EXP might be an MARRAY var.
                                                                     *)
                              EXP)))
          (RETURN ANSWER])
```

(BUILD.COERCE.MLIST

```
  [LAMBDA (EXP MLIST)                                                (* kbr%: "25-Nov-85 17:27")
    (PROG (TYPE ANSWER)
          (SETQ TYPE (fetch (MLIST TYPE) of MLIST))
          (SETQ ANSWER (COND
                           ((NOT (LISTP EXP))
                            EXP)
                           [(EQ (CAR EXP)
                                'LIST)
                            `(LIST ,@(for ITEM in (CDR EXP) collect (BUILD.COERCE ITEM TYPE]
                           [(EQ (CAR EXP)
                                'CONS)
                            `(CONS ,(BUILD.COERCE (CADR EXP)
                                           TYPE)
                                   ,(BUILD.COERCE (CADDR EXP)
                                           MLIST]
                           (T EXP)))
          (RETURN ANSWER])
```

(BUILD.COERCE.EXPLIST

```
  [LAMBDA (EXPLIST MRECORD)                                          (* kbr%: "25-Nov-85 17:27")
                                                                     (* Converts a Mesa explist EXPLIST
                                                                     (ambiguous by itself) into a CREATE TYPE Lisp expression.
                                                                     *)
    (PROG (FIELDLIST ALIGNMENT SETTINGS ANSWER)
          (COND
             ((NOT (type? EXPLIST EXPLIST))
              (RETURN EXPLIST)))
          [COND
             ((NOT (type? MRECORD MRECORD))
              (printout T T MRECORD " not a record" T)               (* Proceed to do the best we can.
                                                                     *)
              [COND
                 ((type? KEYLIST EXPLIST)
                  [SETQ SETTINGS (for ITEM in (fetch (KEYLIST ITEMS) of EXPLIST)
                                    join `(,(fetch (KEYITEM ID) of ITEM)
                                                 _
                                                 ,(fetch (KEYITEM OPTEXP) of ITEM]
                  (RETURN `(create ,MRECORD ,@SETTINGS]
              (RETURN `(,MRECORD ,@(fetch (EXPLIST ITEMS) of EXPLIST]
          (SETQ FIELDLIST (fetch (MRECORD FIELDLIST) of MRECORD))
          (SETQ ALIGNMENT (BUILD.ALIGN FIELDLIST EXPLIST))
          [SETQ SETTINGS (COND
                             [(type? PAIRLIST FIELDLIST)
                              (for PAIRITEM in (fetch (PAIRLIST ITEMS) of FIELDLIST) as ALIGNVALUE in ALIGNMENT
                                 when [NOT (FMEMB ALIGNVALUE '(NIL TRASH]
                                 join `(,(fetch (PAIRITEM ID) of PAIRITEM)
                                        _
                                        ,ALIGNVALUE]
                             [(type? TYPELIST FIELDLIST)
                              (for TYPEITEM in (fetch (TYPELIST ITEMS) of FIELDLIST) as ALIGNVALUE in ALIGNMENT
                                 as I from 1 when [NOT (FMEMB ALIGNVALUE '(NIL TRASH]
                                 join `(,(PACK* 'FIELD I)
                                        _
                                        ,ALIGNVALUE]
                             (T (SHOULDNT]
        EXIT
          [SETQ ANSWER `(create ,(fetch (MRECORD RECORDID) of MRECORD)
                                 ,@SETTINGS]
          (RETURN ANSWER])

(BUILD.ALIGN
  [LAMBDA (FIELDLIST EXPLIST)                                    (* kbr%: "25-Nov-85 17:27")
    (PROG (ANSWER)
          [SETQ ANSWER (COND
                          ((AND (NULL FIELDLIST)
                                (NULL EXPLIST))
                             NIL)
                          ((EQ FIELDLIST 'ANY)
                             (fetch (EXPLIST ITEMS) of EXPLIST))
                          [(type? ORDERLIST EXPLIST)
                             (COND
                                ((type? PAIRLIST FIELDLIST)
                                 (for PAIRITEM in (fetch (PAIRLIST ITEMS) of FIELDLIST) as OPTEXP
                                    in (fetch (ORDERLIST ITEMS) of EXPLIST) collect (BUILD.ALIGN.VALUE
                                                                                      (fetch (PAIRITEM TYPEEXP)
                                                                                         of PAIRITEM)
                                                                                      (fetch (PAIRITEM DEFAULT)
                                                                                         of PAIRITEM)
                                                                                      OPTEXP)))
                                ((type? TYPELIST FIELDLIST)
                                 (for TYPEITEM in (fetch (TYPELIST ITEMS) of FIELDLIST) as OPTEXP
                                    in (fetch (ORDERLIST ITEMS) of EXPLIST) collect (BUILD.ALIGN.VALUE
                                                                                      (fetch (TYPEITEM TYPEEXP)
                                                                                         of TYPEITEM)
                                                                                      (fetch (TYPEITEM DEFAULT)
                                                                                         of TYPEITEM)
                                                                                      OPTEXP)))
                                (T (SHOULDNT]
                          [(type? KEYLIST EXPLIST)
                             (COND
                                ((NOT (type? PAIRLIST FIELDLIST))
                                 (SHOULDNT)))
                             (for PAIRITEM in (fetch (PAIRLIST ITEMS) of FIELDLIST)
                                collect (BUILD.ALIGN.VALUE (fetch (PAIRITEM TYPEEXP) of PAIRITEM)
                                               (fetch (PAIRITEM DEFAULT) of PAIRITEM)
                                               (fetch (KEYITEM OPTEXP) of (ASSOC (fetch (PAIRITEM ID) of PAIRITEM)
                                                                                 (fetch (KEYLIST ITEMS) of EXPLIST]
                          (T (SHOULDNT]
          (RETURN ANSWER])

(BUILD.ALIGN.VALUE
  [LAMBDA (TYPEEXP DEFAULT OPTEXP)                               (* kbr%: "25-Nov-85 17:27")
    (PROG (ANSWER)
          [SETQ ANSWER (OR (COND
                                ((AND (fetch (DEFAULT TRASH) of DEFAULT)
                                      (EQ OPTEXP 'TRASH))
                                 'TRASH))
                             (BUILD.COERCE OPTEXP TYPEEXP)
                             (COPY (fetch (DEFAULT EXP) of DEFAULT]
          (RETURN ANSWER])

(BUILD.ADD.TO.FILECOMS
  [LAMBDA (NAME TYPE)                                            (* kbr%: "25-Nov-85 17:27")
    (PROG (FILECOMSVAR FILECOMS)
          (SETQ FILECOMSVAR BUILD.FILECOMS)
          (SETQ FILECOMS (GETTOPVAL FILECOMSVAR))                (* FILECOMS is reversed at this point.
                                                                     *)
          [COND
             ((AND FILECOMS (EQ (CAR (CAR FILECOMS))
                                 TYPE))
              (NCONC (CAR FILECOMS)
                     (LIST NAME)))
             (T (push FILECOMS (LIST TYPE NAME]
          (SETTOPVAL FILECOMSVAR FILECOMS])

(BUILD.ADD1
  [LAMBDA (EXP)                                                  (* kbr%: "25-Nov-85 17:27")
    (COND
         ((FIXP EXP)
          (ADD1 EXP))
         (T `(ADD1 ,EXP])
```

(**BUILD.CALL**
```
  [LAMBDA (FN EXPLIST)                                    (* kbr%: "25-Nov-85 17:27")
                                                          (* Function call. Cons FN onto front of coerced EXPLIST items.
                                                          *)
    (CONS FN (BUILD.ALIGN (GETPROP FN 'MESA.ARGLIST)
                    EXPLIST])
```

(**BUILD.CHARCODE**
```
  [LAMBDA (CHARCODE)                                      (* kbr%: "25-Nov-85 17:27")
    (PROG (META CONTROL CHAR NAME ANSWER)
          [SETQ NAME (SELECTQ CHARCODE
                         (0 'NULL)
                         (7 'BELL)
                         (8 'BS)
                         (9 'TAB)
                         (10 'LF)
                         (12 'FF)
                         (13 'CR)
                         (27 'ESC)
                         (32 'SPACE)
                         (127 'DEL)
                         (PROGN [COND
                                   ((IGEQ CHARCODE 128)
                                    (SETQ META T)
                                    (SETQ CHARCODE (IDIFFERENCE CHARCODE 128]
                                [COND
                                   ((ILESSP CHARCODE 32)
                                    (SETQ CONTROL T)
                                    (SETQ CHARCODE (IPLUS CHARCODE 32]
                                (SETQ CHAR (MKATOM (CHARACTER CHARCODE)))
                                (COND
                                   ((AND META CONTROL)
                                    (PACK* '%#^ CHAR))
                                   (META (PACK* '%# CHAR))
                                   (CONTROL (PACK* '^ CHAR))
                                   (T CHAR]
          (SETQ ANSWER (LIST 'CHARCODE NAME))
          (RETURN ANSWER])
```

(**BUILD.COND**
```
  [LAMBDA (EXP1 EXP2 EXP3)                                (* kbr%: "25-Nov-85 17:27")
    (PROG (HEAD TAIL ANSWER)
          (SETQ HEAD (CONS EXP1 (BUILD.TAIL EXP2)))
          [SETQ TAIL (COND
                        ((NULL EXP3)
                         NIL)
                        ((AND (LISTP EXP3)
                              (EQ (CAR EXP3)
                                  'COND))
                         (CDR EXP3))
                        (T `((T ,@(BUILD.TAIL EXP3]
          [SETQ ANSWER `(COND
                           ,HEAD
                           ,@TAIL]
          (RETURN ANSWER])
```

(**BUILD.COPY.OF**
```
  [LAMBDA (EXP)                                           (* kbr%: "25-Nov-85 17:27")
    (COND
       ((AND (LISTP EXP)
             (EQ (CAR EXP)
                 'SETQ))
        (CADR EXP))
       (T (COPY EXP])
```

(**BUILD.FETCH**
```
  [LAMBDA (RECORDNAME FIELDNAME DATUM)                    (* kbr%: "25-Nov-85 17:27")
    (PROG (MRECORD ANSWER)
          [SETQ MRECORD (COND
                           (RECORDNAME (BUILD.REFINE.TYPE RECORDNAME))
                           (T (BUILD.ULTIMATE.TYPE DATUM]
          [SETQ ANSWER (COND
                          [(type? MRECORD MRECORD)
                           (SETQ RECORDNAME (fetch (MRECORD RECORDID) of MRECORD))
                           `(fetch (,RECORDNAME ,FIELDNAME) of ,DATUM]
                          (T (printout T T "Bad fetch " RECORDNAME " " FIELDNAME " " DATUM T)
                             (LIST FIELDNAME DATUM]
          (RETURN ANSWER])
```

```
(BUILD.FORCLAUSE.BY
  [LAMBDA (CONTROLID EXP1 EXP2)                                          (* kbr%: "25-Nov-85 17:27")
    `(for ,CONTROLID _ ,EXP1 by ,EXP2])

(BUILD.FORCLAUSE.IN
  [LAMBDA (CONTROLID DIRECTION RANGE)                                    (* kbr%: "25-Nov-85 17:27")
    (PROG (INTERVAL LBOUND UBOUND ANSWER)
          (SETQ INTERVAL (fetch (MRANGE INTERVAL) of RANGE))
          (SETQ LBOUND (fetch (MINTERVAL LBOUND) of INTERVAL))
          (SETQ UBOUND (fetch (MINTERVAL UBOUND) of INTERVAL))
          (SELECTQ (fetch (MINTERVAL KIND) of INTERVAL)
               (CC)
               (CO (SETQ UBOUND (BUILD.SUB1 UBOUND)))
               (OC (SETQ LBOUND (BUILD.ADD1 LBOUND)))
               (OO (SETQ LBOUND (BUILD.ADD1 LBOUND))
                   (SETQ UBOUND (BUILD.SUB1 UBOUND)))
               (SHOULDNT))
          [SETQ ANSWER (COND
                          ((EQ DIRECTION 'DECREASING)
                           `(for ,CONTROLID from ,LBOUND to ,UBOUND by -1))
                          (T `(for ,CONTROLID from ,LBOUND to ,UBOUND]
          (RETURN ANSWER])

(BUILD.FORCLAUSE.THROUGH
  [LAMBDA (RANGE)                                                        (* kbr%: "25-Nov-85 17:27")
    (BUILD.FORCLAUSE.IN 'X NIL RANGE])

(BUILD.IN
  [LAMBDA (EXP RANGE)                                                    (* kbr%: "25-Nov-85 17:28")
    (PROG (INTERVAL EXP2 LPRED UPRED ANSWER)
          (SETQ RANGE (BUILD.REFINE.TYPE RANGE))
          [COND
             ((NOT (type? MRANGE RANGE))
              (printout T T RANGE " not a range." T)
              (RETURN `(in ,RANGE]
          (SETQ INTERVAL (fetch (MRANGE INTERVAL) of RANGE))
          (SELECTQ (fetch (MINTERVAL KIND) of INTERVAL)
               (CC (SETQ LPRED 'IGEQ)
                   (SETQ UPRED 'ILEQ))
               (CO (SETQ LPRED 'IGEQ)
                   (SETQ UPRED 'ILESSP))
               (OC (SETQ LPRED 'IGREATERP)
                   (SETQ UPRED 'ILEQ))
               (OO (SETQ LPRED 'ILESSP)
                   (SETQ UPRED 'IGREATERP))
               (SHOULDNT))
          (SETQ EXP2 (BUILD.COPY.OF EXP))
          [SETQ ANSWER `(AND (,LPRED ,EXP ,(fetch (MINTERVAL LBOUND) of INTERVAL))
                             (,UPRED ,EXP2 ,(fetch (MINTERVAL UBOUND) of INTERVAL]
          (RETURN ANSWER])

(BUILD.ISTYPE
  [LAMBDA (EXP TYPE)                                                     (* kbr%: "25-Nov-85 17:28")
    (PROG (MRECORD RECORDID ANSWER)
          (SETQ MRECORD (BUILD.REFINE.TYPE TYPE))
          (SETQ RECORDID (COND
                            ((type? MRECORD MRECORD)
                             (fetch (MRECORD RECORDID) of MRECORD))
                            (T (printout T T "Bad istype " EXP " " TYPE T)
                               TYPE)))
          [SETQ ANSWER `(type? ,TYPE ,EXP]
          (RETURN ANSWER])

(BUILD.LAMBDA
  [LAMBDA (PAIRLIST BODY)                                                (* kbr%: "25-Nov-85 17:28")
    (PROG (ARGLIST ANSWER)
          [SETQ ARGLIST (for ITEM in (fetch (PAIRLIST ITEMS) of PAIRLIST) collect (BUILD.LOCALVARID
                                                                                   NIL
                                                                                   (fetch (PAIRITEM ID) of ITEM]
          [SETQ ANSWER `(LAMBDA ,ARGLIST
                          ,@(BUILD.TAIL BODY]
          (RETURN ANSWER])

(BUILD.NEW
  [LAMBDA (TYPEEXP INITIALIZATION)                                       (* kbr%: "25-Nov-85 17:28")
    (BUILD.COERCE INITIALIZATION TYPEEXP])

(BUILD.OR
  [LAMBDA (EXPS)                                                    (* kbr%: "25-Nov-85 17:28")
    (COND
       ((NULL EXPS)
        T)
       ((NULL (CDR EXPS))
        (CAR EXPS))
       (T `(OR ,@EXPS])
```

(BUILD.PROG

```
  [LAMBDA (STATEMENTLIST)                                           (* kbr%: "25-Nov-85 17:28")
    (PROG (VARS LAST ANSWER)
          [SETQ ANSWER (APPEND (fetch (SCOPE INITLIST) of BUILD.CURRENT.SCOPE)
                               (BUILD.TAIL (BUILD.PROGN STATEMENTLIST]
          (SETQ VARS (APPEND (fetch (SCOPE VARLIST) of BUILD.CURRENT.SCOPE)
                             (fetch (SCOPE RETURNVARS) of BUILD.CURRENT.SCOPE)))
          [COND
             [(OR VARS (fetch (SCOPE RETURNS) of BUILD.CURRENT.SCOPE)
                  (for EXP in ANSWER thereis (LITATOM EXP)))        (* Local vars, return, or go here. *)
              [COND
                 (ANSWER (SETQ LAST (CAR (LAST ANSWER]
              [COND
                 ([NOT (OR (NULL (fetch (SCOPE RETURNVARS) of BUILD.CURRENT.SCOPE))
                           (AND (LISTP LAST)
                                (FMEMB (CAR LAST)
                                       '(GO RETURN]
                  (SETQ ANSWER (APPEND ANSWER (LIST (BUILD.RETURN]
              (SETQ ANSWER `(PROG ,VARS
                                  ,@ANSWER]
             (T (SETQ ANSWER (BUILD.PROGN ANSWER]
          (RETURN ANSWER])
```

(BUILD.PROGN

```
  [LAMBDA (EXPS)                                                    (* kbr%: "25-Nov-85 17:28")
    (COND
       ((NULL EXPS)
        NIL)
       ((NULL (CDR EXPS))
        (CAR EXPS))
       (T (CONS 'PROGN (for EXP in EXPS join (BUILD.TAIL EXP])
```

(BUILD.REPLACE

```
  [LAMBDA (RECORDNAME FIELDNAME DATUM VALUE)                        (* kbr%: "25-Nov-85 17:28")
    (PROG (MRECORD ANSWER)
          [SETQ MRECORD (COND
                           (RECORDNAME (BUILD.REFINE.TYPE RECORDNAME))
                           (T (BUILD.ULTIMATE.TYPE DATUM]
          [SETQ ANSWER (COND
                          [(type? MRECORD MRECORD)
                           (SETQ RECORDNAME (fetch (MRECORD RECORDID) of MRECORD))
                           `(replace (,RECORDNAME ,FIELDNAME) of ,DATUM with ,VALUE]
                          (T (printout T T "Bad replace " RECORDNAME " " FIELDNAME " " DATUM " " VALUE T)
                             (LIST FIELDNAME DATUM]
          (RETURN ANSWER])
```

(BUILD.RETURN

```
  [LAMBDA (OPTARGS)                                                 (* kbr%: "25-Nov-85 17:28")
                                                                    (* COPY so DEDIT won't get confused by shared structure.
                                                                    *)
    (PROG (SCOPE FN PROCID FIELDLIST EXPLIST ALIGNMENT ANSWER)     (* Get scope of innermost PROC or DO.
                                                                    *)
          (SETQ SCOPE (for SCOPE in (CONS BUILD.CURRENT.SCOPE BUILD.SCOPE.STACK) thereis (fetch (SCOPE ID)
                                                                                                of SCOPE)))
          (replace (SCOPE RETURNS) of SCOPE with T)
          (SETQ FN (fetch (SCOPE ID) of SCOPE))
          [SETQ ALIGNMENT (COND
                             ((EQ FN 'DO)
                              OPTARGS)
                             (OPTARGS (SETQ PROCID (BUILD.PROCID BUILD.PREFIX FN))
                                    [SETQ FIELDLIST (OR (GETPROP PROCID 'MESA.RETURNLIST)
                                                        (PROGN (printout T T "No returnlist for " PROCID "." T)
                                                               'ANY]
                                    (BUILD.ALIGN FIELDLIST OPTARGS))
                             (T (fetch (SCOPE RETURNVARS) of SCOPE]
          [SETQ ANSWER (COND
                          ((NULL ALIGNMENT)
                           (LIST 'RETURN))
                          [(NULL (CDR ALIGNMENT))
                           `(RETURN ,@ALIGNMENT]
                          (T `(RETURN (LIST ,@ALIGNMENT]
          (RETURN ANSWER])
```

(BUILD.SELECTQ

```
  [LAMBDA (CASEHEAD CLAUSES OTHERWISE)                          (* kbr%: "25-Nov-85 17:28")
    (PROG (ID EXP OPTEXP TYPE FN CCLAUSES SCLAUSES ANSWER)
          (SETQ ID (fetch (CASEHEAD ID) of CASEHEAD))
          (SETQ EXP (fetch (CASEHEAD EXP) of CASEHEAD))
          (SETQ OPTEXP (fetch (CASEHEAD OPTEXP) of CASEHEAD))
          (SETQ EXP (OR OPTEXP ID EXP))
          (COND
             ((EQ EXP T)                                        (* Mesa SELECT TRUE FROM statement.
                                                                *)
              (SETQ ANSWER (BUILD.SELECTTRUEFROM CLAUSES OTHERWISE))
              (RETURN ANSWER)))
          (SETQ TYPE (BUILD.ULTIMATE.TYPE EXP))
          (SETQ FN (BUILD.SELECTQ.FN TYPE))
          [for CLAUSE in CLAUSES do (COND
                                       ([for CASETEST in (CAR CLAUSE)
                                           thereis (COND
                                                      ((AND (LISTP CASETEST)
                                                            (FMEMB (CAR CASETEST)
                                                                   '(IN type?]
                                        (push CCLAUSES CLAUSE))
                                       (T (push SCLAUSES CLAUSE]
          (SETQ CCLAUSES (DREVERSE CCLAUSES))
          (SETQ SCLAUSES (DREVERSE SCLAUSES))
          (SETQ CCLAUSES (for CCLAUSE in CCLAUSES collect (BUILD.SELECTQ.CCLAUSE EXP CCLAUSE TYPE)))
          (SETQ SCLAUSES (for SCLAUSE in SCLAUSES collect (BUILD.SELECTQ.SCLAUSE SCLAUSE TYPE)))
          (SETQ ANSWER (COND
                          [SCLAUSES `(,FN ,EXP ,@SCLAUSES ,OTHERWISE]
                          (T OTHERWISE)))
          (SETQ ANSWER (COND
                          [CCLAUSES (COND
                                       [ANSWER `(COND
                                                   ,@CCLAUSES
                                                   (T ,@(BUILD.TAIL ANSWER]
                                       (T `(COND
                                              ,@CCLAUSES]
                          (T ANSWER)))
          (RETURN ANSWER])
```

(BUILD.SELECTQ.FN

```
  [LAMBDA (TYPE)                                                (* kbr%: "25-Nov-85 17:28")
    (COND
       ((EQ TYPE 'CHARACTER)
        'SELCHARQ)
       (T 'SELECTQ])
```

(BUILD.SELECTQ.CCLAUSE

```
  [LAMBDA (EXP CCLAUSE TYPE)                                    (* kbr%: "25-Nov-85 17:28")
    (PROG (EXP2 KEYS TESTS ANSWER)
          (SETQ EXP2 (BUILD.COPY.OF EXP))
          (SETQ KEYS (CAR CCLAUSE))
          [SETQ TESTS (CONS (BUILD.SELECTQ.TEST EXP (CAR KEYS))
                            (for KEY in (CDR KEYS) collect (BUILD.SELECTQ.TEST EXP KEY]
          [COND
             ((NULL (CDR TESTS))
              (SETQ TESTS (CAR TESTS)))
             (T (SETQ TESTS (CONS 'OR TESTS]
          (SETQ ANSWER (CONS TESTS (CDR CCLAUSE)))
          (RETURN ANSWER])
```

(BUILD.SELECTQ.TEST

```
  [LAMBDA (EXP KEY)                                             (* kbr%: "25-Nov-85 17:28")
    (COND
       ((AND (LISTP KEY)
             (EQ (CAR KEY)
                 'IN))
        (BUILD.IN EXP (CADR KEY)))
       ((AND (LISTP KEY)
             (EQ (CAR KEY)
                 'type?))
        KEY)
       (T `(FMEMB ,EXP ',KEY])
```

(BUILD.SELECTQ.SCLAUSE

```
  [LAMBDA (SCLAUSE TYPE)                                        (* kbr%: "25-Nov-85 17:28")
    (PROG (KEYS ANSWER)
          (SETQ KEYS (CAR SCLAUSE))
          (SETQ KEYS (for KEY in KEYS collect (BUILD.SELECTQ.KEY KEY TYPE)))
          [COND
             ((NULL (CDR KEYS))
              (SETQ KEYS (CAR KEYS]
          (SETQ ANSWER (CONS KEYS (CDR SCLAUSE)))
          (RETURN ANSWER])
```

```
(BUILD.SELECTQ.KEY
  [LAMBDA (KEY TYPE)                                                   (* kbr%: "25-Nov-85 17:28")
    (COND
       ((EQ TYPE 'CHARACTER)
        (COND
           [(LISTP KEY)
            (COND
               ((EQ (CAR KEY)
                    'CHARCODE)
                (CADR KEY))
               ((EQ (CAR KEY)
                    'IN)
                (LIST 'IN (LIST (CAR (CADR KEY))
                                (BUILD.SELECTQ.KEY (CADR (CADR KEY))
                                       'CHARACTER)
                                (BUILD.SELECTQ.KEY (CADDR (CADR KEY))
                                       'CHARACTER]
           (T KEY)))
       (T KEY])

(BUILD.SELECTTRUEFROM
  [LAMBDA (CLAUSES OTHERWISE)                                          (* kbr%: "25-Nov-85 17:28")
    (PROG (ANSWER)
          (SETQ CLAUSES (for CLAUSE in CLAUSES collect (BUILD.SELECTTRUEFROM.CLAUSE CLAUSE)))
          (SETQ ANSWER (COND
                          [CLAUSES (COND
                                      [OTHERWISE `(COND
                                                     ,@CLAUSES
                                                     (T ,@(BUILD.TAIL OTHERWISE]
                                      (T `(COND
                                             ,@CLAUSES]
                          (T OTHERWISE)))
          (RETURN ANSWER])

(BUILD.SELECTTRUEFROM.CLAUSE
  [LAMBDA (CLAUSE)                                                     (* kbr%: "25-Nov-85 17:28")
    (CONS (BUILD.OR (CAR CLAUSE))
          (CDR CLAUSE])

(BUILD.SETQ
  [LAMBDA (LHS RHS)                                                    (* kbr%: "25-Nov-85 17:28")
    (PROG (TYPE ANSWER)
          (COND
             ((type? ORDERLIST LHS)
              (SETQ ANSWER (BUILD.SETQ.ORDERLIST LHS RHS))
              (RETURN ANSWER)))
          (SETQ TYPE (BUILD.ULTIMATE.TYPE LHS))
          (SETQ RHS (BUILD.COERCE RHS TYPE))
          [SETQ ANSWER (COND
                          ((NULL LHS)
                           RHS)
                          ((type? MARRAY TYPE)
                           (BUILD.SETQ.ARRAY LHS RHS))
                          [(LISTP LHS)
                           (SELECTQ (CAR LHS)
                               (ELT `(SETA ,(CADR LHS)
                                           ,(CADDR LHS)
                                           ,RHS))
                               (fetch `(replace ,@(CDR LHS) with ,RHS))
                               (NTHCHARCODE `(RPLCHARCODE ,(CADR LHS)
                                                      ,(CADDR LHS)
                                                      ,RHS))
                               (PROGN (printout T "Bad setq " LHS " " RHS)
                                      (COND
                                         [(IEQP (LENGTH LHS)
                                                2)
                                          (COND
                                             [(FIXP (CADR LHS)) (* Guess array access. *)
                                              `(SETA ,(CAR LHS)
                                                     ,(CADR LHS)
                                                     ,RHS]
                                             (T                 (* Guess record access. *)
                                                (BUILD.REPLACE NIL (CAR LHS)
                                                       (CADR LHS)
                                                       RHS]
                                         (T                     (* Guess it could be anything. *)
                                            `(SETQ ,LHS ,RHS]
                          (T `(SETQ ,LHS ,RHS]
          (RETURN ANSWER])

(BUILD.SETQ.ARRAY
  [LAMBDA (LHS RHS)                                                    (* kbr%: "25-Nov-85 17:28")
```

```
                                                                     (* SETQ array LHS. I.e., FILLARRAY.
                                                                     *)
    (PROG (EXPS ANSWER)
          (COND
             ((NOT (type? ORDERLIST RHS))
              (printout T T "Bad setq array " LHS " " RHS T)
              [SETQ ANSWER `(SETQ ,LHS ,RHS]
              (RETURN ANSWER)))
          (SETQ EXPS (for ORDERITEM in (fetch (ORDERLIST ITEMS) of RHS) as I from 0
                        collect (BUILD.SETQ `(ELT ,LHS ,I)
                                       ORDERITEM)))
          (SETQ ANSWER (BUILD.PROGN EXPS))
          (RETURN ANSWER])
```

(**BUILD.SETQ.ORDERLIST**

```
  [LAMBDA (ORDERLIST RHS)                                            (* kbr%: "25-Nov-85 17:28")
                                                                     (* SETQ orderlist ORDERLIST. *)
    (PROG (ORDERITEMS TEMP TEMPPOS EXPS ANSWER)                      (* Get ORDERITEMS *)
          (SETQ ORDERITEMS (fetch (ORDERLIST ITEMS) of ORDERLIST))
          (COND
             ((NULL ORDERITEMS)
              (RETURN RHS))
             ((NULL (CDR ORDERITEMS))
              [SETQ ANSWER (BUILD.SETQ (CAR ORDERITEMS)
                                  `(CAR ,RHS]
              (RETURN ANSWER)))                                      (* Get TEMPorary variable. *)
          (SETQ TEMP (CAR RHS))
          (SETQ TEMPPOS (STRPOS "." TEMP))
          [COND
             (TEMPPOS (SETQ TEMP (SUBATOM TEMP (ADD1 TEMPPOS)
                                        -1]                          (* Get EXPS. *)
          [SETQ EXPS (COND
                        [(ILEQ (LENGTH ORDERITEMS)
                               3)
                         (for ID in ORDERITEMS when ID as ACCESS in '(CAR CADR CADDR)
                            collect (BUILD.SETQ ID `(,ACCESS ,TEMP]
                        (T (for ID in ORDERITEMS when ID collect (BUILD.SETQ ID `(POP ,TEMP]
          [push EXPS `(SETQ ,TEMP ,RHS]                              (* Build PROGN ANSWER. *)
          (SETQ ANSWER (BUILD.PROGN EXPS))
          (RETURN ANSWER])
```

(**BUILD.SUB1**

```
  [LAMBDA (EXP)                                                      (* kbr%: "25-Nov-85 17:28")
    (COND
       ((FIXP EXP)
        (SUB1 EXP))
       (T `(SUB1 ,EXP])
```

(**BUILD.TAIL**

```
  [LAMBDA (EXP)                                                      (* kbr%: "25-Nov-85 17:28")
    (COND
       ((NULL EXP)
        NIL)
       ((AND (LISTP EXP)
             (EQ (CAR EXP)
                 'PROGN))
        (CDR EXP))
       (T (LIST EXP])
)
```

(**BUILD.INIT**)

(RPAQQ **MESATOLISPCOMS**
[;; MESATOLISP -- By Kelly Roach. Lyricized by L. Masinter

```
         (COMS
```

;;; SCAN: reading mesa/cedar files

```
               [INITVARS (SCAN.STRING (CL:MAKE-ARRAY 256 :INITIAL-ELEMENT '#\A :ELEMENT-TYPE 'CL:CHARACTER
                                             :ADJUSTABLE T :FILL-POINTER 0))
                      (SCAN.CHAR NIL)
                      (SCAN.QDOT NIL)
                      (SCAN.BOTH.RESERVED '(! %# %( %) * + %, - %. |..| / %: ; < <= = => > >= @ ABS ALL AND ANY
                                             APPLY ARRAY BASE BEGIN BROADCAST CODE COMPUTED CONTINUE DECREASING
                                             DEFINITIONS DEPENDENT DESCRIPTOR DIRECTORY DO ELSE ENABLE END
                                             ENDCASE ENDLOOP ENTRY ERROR EXIT EXITS EXPORTS FINISHED FIRST FOR
                                             FORK FRAME FREE FROM GO GOTO IF IMPORTS IN INLINE INTERNAL ISTYPE
                                             JOIN LAST LENGTH LOCKS LONG LOOP LOOPHOLE MACHINE MAX MIN MOD
                                             MONITOR MONITORED NARROW NEW NILL NOT NOTIFY NULL OF OPEN OR ORD
                                             ORDERED OVERLAID PACKED POINTER PORT PRED PRIVATE PROC PROCEDURE
                                             PROCESS PROGRAM PUBLIC READONLY RECORD REJECT RELATIVE REPEAT
                                             RESTART RESUME RETRY RETURN RETURNS SELECT SEQUENCE SHARES SIGNAL
```

```
                                  SIZE START STATE STOP SUCC THEN THROUGH TO TRANSFER TRASH TYPE
                                  UNCOUNTED UNTIL USING VAL VAR WAIT WHILE WITH ZONE %[ %] ^ _ { %| }
                                  ~))
               (SCAN.CEDAR.RESERVED '(CEDAR CHECKED CONS LIST PAINTED REF SAFE TRUSTED UNCHECKED UNSAFE))
               (SCAN.MESA.RESERVED '(RESIDENT]
        (FNS SCAN.INIT SCAN.START SCAN.TEST SCAN.TESTFILE SCAN.OPENSTREAM SCAN.TOKEN SCAN.NUMBER
             SCAN.ACCEPT SCAN.APPENDDECIMAL SCAN.APPENDOCTAL SCAN.APPENDHEX SCAN.APPENDTOSCALE
             SCAN.VALIDFRACTION SCAN.DECIMAL SCAN.OCTAL SCAN.OCTALCHAR SCAN.HEX SCAN.FLOATING SCAN.ESCAPE)
        (P (SCAN.INIT)))
    (COMS                                                          ; PARSE *
        [INITVARS (PARSE.FILELST NIL)
               (PARSE.STREAM NIL)
               (PARSE.FILECOMS NIL)
               (PARSE.LANGUAGE NIL)
               (PARSE.DIRLST NIL)
               (PARSE.CLASS NIL)
               (PARSE.ATOM NIL)
               (PARSE.CLASS2 NIL)
               (PARSE.ATOM2 NIL)
               (PARSE.CASEHEAD.FIRST '(WITH SELECT))
               (PARSE.DEFHEAD.FIRST '(DEFINITIONS))
               (PARSE.DEPENDENT.FIRST '(MACHINE))
               (PARSE.DOTEST.FIRST '(UNTIL WHILE))
               (PARSE.FORCLAUSE.FIRST '(FOR THROUGH))
               (PARSE.HEAP.FIRST '(UNCOUNTED))
               (PARSE.INTERVAL.FIRST '(%( %[))
               (PARSE.OPTRELATION.FIRST '(%# < <= = > >= IN NOT ~))
               (PARSE.ORDERED.FIRST '(ORDERED))
               (PARSE.ORDERLIST.FOLLOW '(! ; END %] }))
               (PARSE.PACKED.FIRST '(PACKED))
               (PARSE.PREFIXOP.FIRST '(ABS BASE LENGTH LONG MAX MIN ORD PRED SUCC))
               (PARSE.PROGHEAD.FIRST '(MONITOR PROGRAM RESIDENT))
               (PARSE.QUALIFIER.FIRST '(%. %[ ^))
               (PARSE.RANGE.FOLLOW '(! %) %, |..| %: ; => AND DO ELSE END ENDCASE ENDLOOP EXITS FINISHED
                                  FROM NULL OR REPEAT SELECT THEN TRASH UNTIL WHILE %] }))
               (PARSE.TRANSFER.FIRST '(BROADCAST ERROR JOIN NOTIFY RESTART RETURN SIGNAL START TRANSFER))
               (PARSE.TRANSFERMODE.FIRST '(ERROR PORT PROCESS PROGRAM SIGNAL))
               (PARSE.TRANSFEROP.FIRST '(ERROR FORK JOIN NEW SIGNAL START))
               (PARSE.TYPECONS.FIRST '(%( ARRAY BASE DESCRIPTOR ERROR FRAME LONG MACHINE MONITORED ORDERED
                                    PACKED POINTER PORT PROC PORCEDURE PROCESS PROGRAM RECORD SIGNAL
                                    UNCOUNTED VAR %[ {))
               (PARSE.TYPEOP.FIRST '(FIRST LAST NILL))
               (PARSE.VARIANTPART.FIRST '(PACKED SELECT SEQUENCE))
               (PARSE.CATCHLIST.FOLLOW '(END %] }))
               (PARSE.CONTROLID.FOLLOW '(DECREASING IN _))
               (PARSE.DECLIST.FOLLOW '(; END }))
               (PARSE.DEFAULTOPT.FOLLOW '(%, ; END %] }))
               (PARSE.EXITLIST.FOLLOW '(END ENDLOOP FINISHED }))
               (PARSE.MODULELIST.FOLLOW '(IEQP EXPORTS SHARES))
               (PARSE.OPTARGS.FOLLOW '(; ELSE END ENDCASE ENDLOOP EXITS FINISHED REPEAT %] }))
               (PARSE.OPTEXP.FOLLOW '(! %, ; END FROM %] }))
               (PARSE.SCOPE.FOLLOW '(END EXITS }))
               (PARSE.STATEMENTLIST.FOLLOW '(END ENDLOOP EXITS REPEAT }))
               (PARSE.TYPEEXP.FOLLOW '(! %, ; = => DECREASING END EXPORTS FROM IMPORTS IN OF SHARES %] _ }
                                    ))
               (PARSE.PREDEFINED.TYPES '(ATOM BOOL BOOLEAN CARDINAL CHAR CHARACTER CONDITION INT INTEGER
                                        MDSZone MONITORLOCK NAT REAL STRING StringBody UNSPECIFIED
                                        WORD))
               (PARSE.RELOPS (LIST '= '%# '< '<= '> '>=))
               (PARSE.ADDOPS (LIST '+ '-))
               (PARSE.MULTOPS (LIST '* '/ 'MOD))
               (PARSE.TRANSFEROPS '(SIGNAL ERROR START JOIN NEW FORK))
               (PARSE.PREFIXOPS '(LONG ABS PRED SUCC ORD MIN MAX BASE LENGTH))
               (PARSE.TYPEOPS '(FIRST LAST NILL))
               (PARSE.NOTS '(~ NOT]
        (RECORDS PARSERSTATE MINTERVAL MRANGE MRELATIVE MPAINTED MENUMERATED MRECORD MVAR MARRAY
               MDESCRIPTOR MFRAME MREF MLIST PAIRITEM DEFAULT TYPELIST TYPEITEM MPOINTER CASEHEAD BINDITEM
               KEYITEM FIELDLIST PAIRLIST ORDERLIST KEYLIST EXPLIST)
        (FNS PARSE.MESA PARSE.CEDAR PARSE.FILE PARSE.GET.STATE PARSE.SET.STATE PARSE.BIN PARSE.VARID
             PARSE.SMURF PARSE.THISIS.MESA PARSE.THISIS.CEDAR PARSE.MODULE PARSE.INCLUDEITEM
             PARSE.INCLUDECHECK PARSE.SEADIRT PARSE.PROGHEAD PARSE.RESIDENT PARSE.SAFE PARSE.DEFHEAD
             PARSE.TILDE PARSE.DEFINITIONS PARSE.DEFBODY PARSE.LOCKS PARSE.LAMBDA PARSE.MODULEITEM
             PARSE.DECLARATION PARSE.PUBLIC PARSE.ENTRY PARSE.IDLIST PARSE.IDENTLIST PARSE.POSITION
             PARSE.OPTBITS PARSE.INTERVAL PARSE.TYPEEXP.HERE PARSE.TYPEEXP PARSE.RANGE PARSE.TYPEAPPL
             PARSE.TYPEAPPL.CONT PARSE.TYPEID PARSE.TYPEID.CONT PARSE.TYPECONS PARSE.TYPECONS1
             PARSE.TYPECONS.CONT PARSE.TYPECONS.RANGE PARSE.TYPECONS.RELATIVE PARSE.TYPECONS.PAINTED
             PARSE.TYPECONS2 PARSE.TYPECONS.INTERVAL PARSE.TYPECONS.DEPENDENT PARSE.TYPECONS.ENUMERATED
             PARSE.TYPECONS.RECORD PARSE.TYPECONS.ORDERED PARSE.TYPECONS.VAR PARSE.TYPECONS.PACKED
             PARSE.TYPECONS.DESCRIPTOR PARSE.TYPECONS.SAFE PARSE.TYPECONS.HEAP PARSE.TYPECONS.LONG
             PARSE.TYPECONS.FRAME PARSE.TYPECONS.REF PARSE.TYPECONS.LIST PARSE.IDENT PARSE.ELEMENT
             PARSE.MONITORED PARSE.DEPENDENT PARSE.RECLIST PARSE.VARIANTPAIR PARSE.PAIRITEM
             PARSE.DEFAULTOPT PARSE.VARIANTPART PARSE.VCASEHEAD PARSE.TAGTYPE PARSE.VARIANTITEM
             PARSE.TYPELIST PARSE.TYPEITEM PARSE.POINTERTYPE PARSE.TRANSFERMODE PARSE.INITIALIZATION
             PARSE.INITVALUE PARSE.CHECKED PARSE.CODELIST PARSE.STATEMENT PARSE.STATEMENT1
             PARSE.STATEMENT2 PARSE.STATEMENT.CASEHEAD PARSE.STATEMENT.FORCLAUSE PARSE.STATEMENT.RETURN
             PARSE.STATEMENT.TRANSFER PARSE.STATEMENT.LBRACKET PARSE.STATEMENT.IF PARSE.BLOCK PARSE.SCOPE
             PARSE.BINDITEM PARSE.EXITS PARSE.CASESTMTITEM PARSE.CASEEXPITEM PARSE.EXITITEM PARSE.CASETEST
```

```
                PARSE.CONTROLID PARSE.FORCLAUSE PARSE.DIRECTION PARSE.DOTEST PARSE.DOEXIT PARSE.ENABLES
                PARSE.CATCHLIST PARSE.CATCHCASE PARSE.OPTARGS PARSE.TRANSFER PARSE.KEYITEM PARSE.OPTEXP
                PARSE.EXP PARSE.EXP1 PARSE.EXP2 PARSE.EXP.TRANSFEROP PARSE.EXP.IF PARSE.EXP.CASEHEAD
                PARSE.EXP.LHS PARSE.EXP.LBRACKET PARSE.EXP.ERROR PARSE.EXP.DISJUNCT PARSE.DISJUNCT
                PARSE.CONJUNCT PARSE.NEGATION PARSE.RELATION PARSE.SUM PARSE.PRODUCT PARSE.OPTRELATION
                PARSE.RELATIONTAIL PARSE.RELOP PARSE.ADDOP PARSE.MULTOP PARSE.FACTOR PARSE.PRIMARY PARSE.ATOM
                PARSE.PRIMARY.NIL PARSE.PRIMARY.LBRACKET PARSE.PRIMARY.PREFIXOP PARSE.PRIMARY.VAL
                PARSE.PRIMARY.ALL PARSE.PRIMARY.NEW PARSE.PRIMARY.TYPEOP PARSE.PRIMARY.SIZE
                PARSE.PRIMARY.ISTYPE PARSE.PRIMARY.AT PARSE.PRIMARY.DESCRIPTOR PARSE.PRIMARY.CONS
                PARSE.PRIMARY.LIST PARSE.PRIMARY.LHS PARSE.PRIMARY.LHS.NEW PARSE.PRIMARY.LHS.CONS
                PARSE.PRIMARY.LHS.LIST PARSE.QUALIFIER PARSE.LHS PARSE.QUALIFIER.HERE PARSE.OPTCATCH
                PARSE.TRANSFEROP PARSE.PREFIXOP PARSE.TYPEOP PARSE.DESCLIST PARSE.DIRECTORY PARSE.IMPORTS
                PARSE.POINTERPREFIX PARSE.EXPORTS PARSE.FIELDLIST PARSE.USING PARSE.CATCHHEAD PARSE.DECLIST
                PARSE.PAIRLIST PARSE.VARIANTLIST PARSE.ORDERLIST PARSE.LHSLIST PARSE.INCLUDELIST
                PARSE.MODULELIST PARSE.ELEMENTLIST PARSE.BINDLIST PARSE.STATEMENTLIST PARSE.CASESTMTLIST
                PARSE.CASELABEL PARSE.EXITLIST PARSE.KEYLIST PARSE.CASEEXPLIST PARSE.EXPLIST PARSE.OPEN
                PARSE.CLASS PARSE.CASEHEAD PARSE.READONLY PARSE.ORDERED PARSE.BASE PARSE.PACKED PARSE.HEAP
                PARSE.INLINE PARSE.ARGUMENTS PARSE.INTERFACE PARSE.SHARES PARSE.DEFAULT PARSE.OPTSIZE
                PARSE.BOUNDS PARSE.LENGTH PARSE.INDEXTYPE PARSE.ELSEPART PARSE.OTHERPART PARSE.FREE
                PARSE.CATCHANY PARSE.NOT PARSE.NEW PARSE.OPTTYPE PARSE.ARGLIST PARSE.RETURNLIST))
      (COMS ;; BUILD

           [INITVARS (BUILD.NEXT.SCOPE NIL)
                  (BUILD.CURRENT.SCOPE NIL)
                  (BUILD.SCOPE.STACK NIL)
                  (BUILD.PREFIX NIL)
                  (BUILD.FILECOMS NIL)
                  (BUILD.BOOLEAN.FNS '(AND OR NOT type? IGREATERP ILESSP IGEQ ILEQ IEQP ZEROP MINUSP EVENP
                                           ODDP FGREATERP FLESSP FEQP GREATERP LESSP GEQ LEQ))
                  (BUILD.CARDINAL.FNS '(ADD1 CHARCODE FIX GCD IDIFFERENCE IMAX IMIN IMINUS IMOD IPLUS
                                             IQUOTIENT IREMAINDER ITIMES LOGAND LOGNOT LOGOR LOGXOR
                                             NTHCHARCODE SUB1))
                  (BUILD.MIXED.FNS '(ABS DIFFERENCE EXPT MAX MIN MINUS MOD PLUS QUOTIENT REMAINDER TIMES))
                  (BUILD.REAL.FNS '(ANTILOG ARCCOS ARCSIN ARCTAN ARCTAN2 COS FDIFFERENCE FLOAT FMAX FMIN
                                         FMINUS FMOD FPLUS FQUOTIENT FREMAINDER FTIMES LOG SIN SQRT TAN))
                  (BUILD.QUALIFY.WORDS '(FREE NEW SIZE))
                  [BUILD.CARDINAL.ARITHOP.ALIST (LIST (CONS '= 'IEQP)
                                                      (CONS '%# 'IEQP)
                                                      (CONS '< 'ILESSP)
                                                      (CONS '<= 'ILEQ)
                                                      (CONS '> 'IGREATERP)
                                                      (CONS '>= 'IGEQ)
                                                      (CONS '+ 'IPLUS)
                                                      (CONS '- 'IDIFFERENCE)
                                                      (CONS '* 'ITIMES)
                                                      (CONS '/ 'IQUOTIENT)
                                                      (CONS '0- 'IMINUS)
                                                      (CONS 'MAX 'IMAX)
                                                      (CONS 'MIN 'IMIN)
                                                      (CONS 'MOD 'IMOD]
                  [BUILD.MIXED.ARITHOP.ALIST (LIST (CONS '= 'EQP)
                                                   (CONS '%# 'EQP)
                                                   (CONS '< 'LESSP)
                                                   (CONS '<= 'GREATERP)
                                                   (CONS '> 'GREATERP)
                                                   (CONS '>= 'LESSP)
                                                   (CONS '+ 'PLUS)
                                                   (CONS '- 'DIFFERENCE)
                                                   (CONS '* 'TIMES)
                                                   (CONS '/ 'QUOTIENT)
                                                   (CONS '0- 'MINUS)
                                                   (CONS 'MAX 'MAX)
                                                   (CONS 'MIN 'MIN)
                                                   (CONS 'MOD 'IMOD]
                  [BUILD.REAL.ARITHOP.ALIST (LIST (CONS '= 'FEQP)
                                                  (CONS '%# 'FEQP)
                                                  (CONS '< 'FLESSP)
                                                  (CONS '<= 'FGREATERP)
                                                  (CONS '> 'FGREATERP)
                                                  (CONS '>= 'FLESSP)
                                                  (CONS '+ 'FPLUS)
                                                  (CONS '- 'FDIFFERENCE)
                                                  (CONS '* 'FTIMES)
                                                  (CONS '/ 'FQUOTIENT)
                                                  (CONS '0- 'FMINUS)
                                                  (CONS 'MAX 'FMAX)
                                                  (CONS 'MIN 'FMIN)
                                                  (CONS 'MOD 'IMOD]
                  (BUILD.CARDINAL.TYPES '(CARDINAL CHAR CHARACTER INT INTEGER NAT WORD]
           (RECORDS SCOPE)
           (FNS BUILD.INIT BUILD.PUSH.SCOPE BUILD.POP.SCOPE BUILD.GC.SCOPE BUILD.STORE.EXPORTS
                BUILD.STORE.IDENTLIST BUILD.STORE.INTERFACES BUILD.STORE.INTERFACE BUILD.STORE.OPEN
                BUILD.STORE.USING BUILD.INITIALIZATION BUILD.INITIALIZE.VARS BUILD.INITIALIZE.VAR
                BUILD.INITIALIZE.FN BUILD.INITIALIZE.RECORD BUILD.RECORD BUILD.TYPE BUILD.STORE.ARGLIST
                BUILD.STORE.RETURNLIST BUILD.STORE.PAIRLIST BUILD.STORE.PAIRITEM BUILD.STORE.VARLIST BUILD.ID
                BUILD.FIELDID BUILD.PROCID BUILD.RECORDID BUILD.TYPEID BUILD.VARID BUILD.LOCALVARID
                BUILD.GLOBALVARID BUILD.ULTIMATE.TYPE BUILD.REFINE.TYPE BUILD.IMMEDIATE.TYPE
```

```
                   BUILD.LOOKUP.TYPE BUILD.LOOKUP BUILD.TYPEATOM BUILD.QUALIFY BUILD.QUALIFY.PREFIXOP
                   BUILD.QUALIFY.TYPEOP BUILD.QUALIFY.EXPLIST BUILD.QUALIFY.ID BUILD.ARITH.EXP1 BUILD.ARITH.EXP2
                   BUILD.ARITH.EXP* BUILD.ARITH.ADD1SUB1 BUILD.COERCE.ARITHOP BUILD.STRONGEST.TYPE.AMONG
                   BUILD.STRONGEST.TYPE BUILD.COERCE BUILD.COERCE.MARRAY BUILD.COERCE.MLIST BUILD.COERCE.EXPLIST
                   BUILD.ALIGN BUILD.ALIGN.VALUE BUILD.ADD.TO.FILECOMS BUILD.ADD1 BUILD.CALL BUILD.CHARCODE
                   BUILD.COND BUILD.COPY.OF BUILD.FETCH BUILD.FORCLAUSE.BY BUILD.FORCLAUSE.IN
                   BUILD.FORCLAUSE.THROUGH BUILD.IN BUILD.ISTYPE BUILD.LAMBDA BUILD.NEW BUILD.OR BUILD.PROG
                   BUILD.PROGN BUILD.REPLACE BUILD.RETURN BUILD.SELECTQ BUILD.SELECTQ.FN BUILD.SELECTQ.CCLAUSE
                   BUILD.SELECTQ.TEST BUILD.SELECTQ.SCLAUSE BUILD.SELECTQ.KEY BUILD.SELECTTRUEFROM
                   BUILD.SELECTTRUEFROM.CLAUSE BUILD.SETQ BUILD.SETQ.ARRAY BUILD.SETQ.ORDERLIST BUILD.SUB1
                   BUILD.TAIL)
              (P (BUILD.INIT)))
        (DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS (ADDVARS (NLAMA)
                                                                              (NLAML)
                                                                              (LAMA PARSE.BIN PARSE.FILE
                                                                                    PARSE.CEDAR])

(DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILERVARS

(ADDTOVAR NLAMA )

(ADDTOVAR NLAML )

(ADDTOVAR LAMA PARSE.BIN PARSE.FILE PARSE.CEDAR)
)

(PUTPROPS MESATOLISP COPYRIGHT ("Xerox Corporation" 1985 1987))
```

FUNCTION INDEX

VARIABLE INDEX

RECORD INDEX