(1) *listByWith*[delim][item] **:** item **(** delim item **)* ;**

(2) *listBy*[delim][item] **:** item **(** delim item **)* ;**

(3) *listWith*[item][f][x] **:** *listByWith*$_{(1:1)}$[**“,”**][item][f][x] **;**

(4) *list* **:** *listBy*$_{(2:1)}$[**“,”**] **;**

(5) *list0ByWith*[delim][item] **:** item **(** delim item **)*** | **empty ;**

(6) *list0By*[delim][item] **:** *listBy*$_{(2:1)}$[delim][item] | **empty ;**

(7) *list0With*[item][f][x] **:** *list0ByWith*$_{(5:1)}$[**“,”**][item][f][x] **;**

(8) *list0* **:** *list0By*$_{(6:1)}$[**“,”**] **;**

(9) *located*[item] **:** item **;**

(10) *ident* **:** *located*$_{(9:2)}$[**IDENT**] **;**

(11) *literal* **:** *located*$_{(9:2)}$[**LITERAL**] **;**

(12) *sequence*[item] **:** *listBy*$_{(2:1)}$[**";"**][item] **;**

(13) *compositeType*[expr][typ] **:**
arrayType(14:4)[expr][typ]
| *recordType*(15:4)[typ]
| *procedureType*(17:4)[typ]
| *ident*(10:3)
;

(14) *arrayType*[expr][typ] **: ARRAY** [expr] **OF** typ **;**

(15) *recordType*[typ] **: RECORD** *sequence*(12:3)[*fieldList*(16:4)[typ]] **END ;**

(16) *fieldList*[typ] **:** *list*(4:1)[*ident*(10:3)] **":"** typ **;**

(17) *procedureType*[typ] **:**
PROCEDURE [*formalParameters*(29:10)[typ]] [**":"** typ]
;

(18) *pointerType*[typ] **:** **“^”** typ **;**

(19) *constantDeclaration*[expr] **: CONST (** *ident*(10:3) **“=”** expr **“;”** **)+ ;**

```
(20) extendedStatement[expr][stmt] :
        forStatement(21:7)[expr][stmt]
      | caseStatement(23:7)[expr][stmt]
      ;

(21) forStatement[expr][stmt] :
      FOR
         ident(10:3)
         ":="
         expr
         ( TO | DOWNTO )
         expr
         [ STEP expr ]
         DO
         sequence(12:3)[stmt]
         END
      ;

(22) caseCondition : literal(11:3) [ ".." literal(11:3) ] ;

(23) caseStatement[expr][stmt] :
      CASE
         expr
         OF
         ( caseCondition(22:7) ":" sequence(12:3)[stmt] )+
         [ ELSE sequence(12:3)[stmt] ]
         END
      ;
```

(24) *moduleDefinition*[declarations][statement] **:**
MODULE
ident(10:3)
";"
declarations
[**BEGIN** *sequence*(12:3)[statement]]
moduleEnd(25:8)[*name*(??:??)]
;

(25) *moduleEnd* **: END expectedName "." ;**

(26) *primitiveType* **: INTEGER** | **BOOLEAN ;**

```
(27) procedureHeading[typ] :
       PROCEDURE
         ident(10:3)
         [ formalParameters(29:10)[typ] ]
         [ ":" typ ]
       ;

(28) parameterSection[typ] : [ VAR ] list(4:1)[ident(10:3)] ":" typ ;

(29) formalParameters[typ] :
       "(" [ sequence(12:3)[parameterSection(28:10)[typ]] ] ")"
       ;

(30) procedureBody[decl][stmt] : decl [ BEGIN sequence(12:3)[stmt] ] ;

(31) procedureDeclaration[typ][decl][stmt] :
       procedureHeading(27:10)[typ]
         ";"
         procedureBody(30:10)[decl][stmt]
         procedureEnd(32:10)[name(??:??)]
       ;

(32) procedureEnd : END expectedName ";" ;
```

```
(33) factor[reference] :
        located(9:2)[literal(11:3)]
      | located(9:2)[TRUE]
      | located(9:2)[FALSE]
      | located(9:2)["(" expression(39:11)[reference] ")"]
      | located(9:2)["~" factor(33:11)[reference]]
      | located(9:2)[reference]
      ;

(34) term[reference] :
      located(9:2)[factor(33:11)[reference] ( multiplicative(35:11) factor(33:11)[reference]
)*]
      ;

(35) multiplicative : "*" | DIV | MOD | "&" ;

(36) simpleExpression[reference] :
      located(9:2)[
          [ unary(38:11) ]
            term(34:11)[reference]
            ( additive(37:11) term(34:11)[reference] )*
      ]
      ;

(37) additive : "+" | "-" | OR ;

(38) unary : "+" | "-" ;

(39) expression[reference] :
      located(9:2)[
          simpleExpression(36:11)[reference]
              [ comparison(40:11) simpleExpression(36:11)[reference] ]
      ]
      ;

(40) comparison : "=" | "#" | "<" | "<=" | ">" | ">=" ;
```

```
(41) simpleStatement[ref][expr][stmt] :
         assignment(42:12)[ref][expr]
       | ifStatement(43:12)[expr][stmt]
       | whileStatement(44:12)[expr][stmt]
       | callStatement(45:12)[ref][expr]
       ;

(42) assignment[ref][expr] : ref ":=" expr ;

(43) ifStatement[expr][stmt] :
       IF
           expr
           THEN
           sequence(12:3)[stmt]
           ( ELSIF expr THEN sequence(12:3)[stmt] )*
           [ ELSE sequence(12:3)[stmt] ]
           END
       ;

(44) whileStatement[expr][stmt] : WHILE expr DO sequence(12:3)[stmt] END
;

(45) callStatement[ref][expr] : ref [ "(" list(4:1)[expr] ")" ] ;
```

(46) *typeDeclaration*[typ] **: TYPE (** *ident*(10:3) **“=”** typ **“;”** **)+ ;**

(47) *variableDeclaration*[typ] **:** **VAR** **(** *list*$_{(4:1)}$[*ident*$_{(10:3)}$] **":"** typ **";"** **)+** **;**

```
(48) reference :
       located(9:2)[“”* ident(10:3) selector(49:15)* | NEW extendedType(51:15)]
       ;

(49) selector :
       located(9:2)[
             ( “.” ident(10:3) )
           | ( “[” expr(50:15) “]” )
           | ( “(” list0(8:1)[expr(50:15)] “)” )
           | “”
       ]
       ;

(50) expr : expression(39:11)[reference(48:15)] ;

(51) extendedType :
         primitiveType(26:9)
       | compositeType(13:4)[expr(50:15)][extendedType(51:15)]
       | pointerType(18:5)[extendedType(51:15)]
       ;

(52) declarations :
       [ constantDeclaration(19:6)[expr(50:15)] ]
         [ typeDeclaration(46:13)[extendedType(51:15)] ]
         [ variableDeclaration(47:14)[extendedType(51:15)] ]
         procedureDeclaration(31:10)[extendedType(51:15)][declarations(52:15)][statement(53:15)]*
       ;

(53) statement :
         simpleStatement(41:12)[reference(48:15)][expr(50:15)][statement(53:15)]
       | extendedStatement(20:7)[expr(50:15)][statement(53:15)]
       | FREE expr(50:15)
       ;

(54) level5 : moduleDefinition(24:8)[declarations(52:15)][statement(53:15)] ;
```