program ::= program-heading block "."

program-heading ::= **program** identifier ";"

block ::= declaration-part statement-part

declaration-part ::=
[type-definition-part]
[variable-declaration-part]
procedure-and-function-declaration-part

type-definition-part ::= **type** type-definition ";" { type-definition ";" }

type-definition ::= identifier "=" type

variable-declaration-part ::= **var** variable-declaration ";" { variable-
declaration ";" }

variable-declaration ::= identifier-list ":" type

procedure-and-function-declaration-part ::= { (procedure-declaration |
function-declaration) ";" }

procedure-declaration ::= procedure-heading ";" procedure-body

procedure-body ::= block

function-declaration ::= function-heading ";" function-body

function-body ::= block

statement-part ::= **begin** statement-sequence **end**

Procedure and Function Definitions

procedure-heading ::= **procedure** identifier [formal-parameter-list]

function-heading ::= **function** identifier [formal-parameter-list] ":"
result-type

result-type ::= type-identifier

formal-parameter-list ::= "(" formal-parameter-section { ";" formal-
parameter-section } ")"

formal-parameter-section ::= value-parameter-section

value-parameter-section ::= identifier-list ":" parameter-type

parameter-type ::= type-identifier | conformant-array-schema

conformant-array-schema ::= array-schema array-schema ::= **array** "[" bound-specification "]" **of** (type-identifier | conformant-array-schema)

bound-specification ::= identifier ".." identifier ":" ordinal-type-identifier

ordinal-type-identifier ::= type-identifier

Statements

statement-sequence ::= statement { ";" statement }

statement ::= (simple-statement | structured-statement)

simple-statement ::= [assignment-statement | procedure-statement]

assignment-statement ::= variable ":=" expression

procedure-statement ::= procedure-identifier [actual-parameter-list]

structured-statement ::=
compound-statement |
repetitive-statement |
conditional-statement

compound-statement ::= **begin** statement-sequence **end**

repetitive-statement ::= while-statement

while-statement ::= **while** expression **do** statement

conditional-statement ::= if-statement

if-statement ::= **if** expression **then** statement [**else** statement]

actual-parameter-list ::= "(" actual-parameter { "," actual-parameter } ")"

actual-parameter ::=
actual-value |
actual-variable

actual-value ::= expression

Expressions

Expression ::= simple-expression [relational-operator simple-expression]

simple-expression ::= [sign] term { addition-operator term }
term ::= factor { multiplication-operator factor }

factor ::=

variable |
number |
constant-identifier |
bound-identifier |
function-designator |
"(" expression ")" |
not factor

relational-operator ::= "=" | "<>" | "<" | "<=" | ">" | ">="

addition-operator ::= "+" | "-" | or

multiplication-operator ::= "*" | "/" | div | mod | and

variable ::= entire-variable | component-variable | referenced-variable

entire-variable ::= variable-identifier | field-identifier

component-variable ::= indexed-variable | field-designator | file-buffer}

indexed-variable ::= array-variable "[" expression-list "]"

field-designator ::= record-variable "." field-identifier

function-designator ::= function-identifier [actual-parameter-list]

<u>Types</u>

Type ::= simple-type | structured-type | type-identifier

simple-type ::= subrange-type | enumerated-type

enumerated-type ::= "(" identifier-list ")"
subrange-type ::= lower-bound ".." upper-bound
lower-bound ::= constant
upper-bound ::= constant

structured-type ::= array-type | record-type
array-type ::= **array** "[" index-type { "," index-type } "]" **of** element-type

index-type ::= simple-type
element-type ::= type
record-type ::= **record** field-list **end**