```
actual-parameter = expression | variable-access |
   procedure-identifier | function-identifier

   actual-parameter-list = '(' actual-parameter { ',' actual-parameter } ')'

   adding-operator = '+' | '-' | 'or' | 'or_else' | 'xor'

   array-type = 'array' [ index-type-list ']' 'of' component-type

   array-variable = variable-access

   assignment-statement = assignment-statement-lhs ':=' expression

   assignment-statement-lhs = variable-access | function-identifier |
property-designator

   base-type = ordinal-type

   binary-digit = '0' | '1'

   binary-digit-sequence = binary-digit { binary-digit }

   binary-integer = '%' binary-digit-sequence

   block = declarative-part statement-part

   boolean-expression = expression

   buffer-variable = file-variable '^' | file-variable '@'

   c-string-type = 'cstring' [ max-string-length ]

   case-body = case-list-elements [ [ ';' ] case-statement-completer ] |
                     case-statement-completer

   case-constant = ordinal-constant

   case-constant-list = case-specifier { ',' case-specifier }

   case-index = ordinal-expression

   case-list-element = case-constant-list ':' statement

   case-list-elements = case-list-element { ';' case-list-element }

   case-specifier = case-constant [ '..' case-constant ]

   case-statement = 'case' case-index case-body [ ';' ] 'end'

   case-statement-completer = ( 'otherwise' | 'else' ) statement-sequence

   character-code = digit-sequence

   character-literal = ''' string-element-one ''' |
```

```
                        '"' string-element-two '"' |
                        '#' character-code

  component-type = type-denoter

  component-variable = indexed-variable | field-designator

  compound-statement = 'begin' statement-sequence 'end'

  conditional-statement = if-statement | case-statement

  constant = [ sign ] integer-number |
             [ sign ] real-number |
             [ sign ] constant-identifier |
                      character-literal |
                      string-literal

  constant-definition = identifier '=' constant

  constant-definition-group = 'const' constant-definition ';' { constant-
definition ';' }

  constant-identifier = identifier

  control-variable = entire-variable

  decimal-integer = digit-sequence

  declaration-group =
          label-declaration-group |
          constant-definition-group |
          type-definition-group |
          variable-declaration-group |
          function-declaration  |
          procedure-declaration

  declarative-part = { declaration-group }

  digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

  digit-sequence = digit { digit }

  directive = forward-directive | external-directive

  dllname = string-literal

  domain-type = type-identifier

  else-part = 'else' statement

  empty-statement =

  empty-string = "'" | '""'
```

```
  entire-variable = variable-identifier

  enumerated-constant = identifier

  enumerated-constant-list = enumerated-constant { ',' enumerated-
constant }

  enumerated-type = '(' enumerated-constant-list ')'

  exponent = 'e'

  expression = shift-expression [ relational-operator shift-expression ]

  external-directive = 'external' dllname [ 'name' '=' name ] [ 'stdcall' |
'cdecl' ]

  factor = [ sign ] unsigned-constant |
           [ sign ] variable-access |
           [ sign ] '(' expression ')' |
           [ sign ] function-designator |
           [ sign ] function-method-designator |
           [ sign ] 'not' factor |
           set-constructor

  field-designator = record-variable '.' field-specifier | field-designator-
identifier

  field-designator-identifier = identifier

  field-identifier = identifier

  field-list = [
      fixed-part ';' variant-part [ ';' ] |
      fixed-part [ ';' ] |
      variant-part [ ';' ] |
    ]

  field-specifier = field-identifier

  file-type = 'file' 'of' component-type

  file-variable = variable-access

  final-value = ordinal-expression

  fixed-part = record-section { ';' record-section }

  for-statement = 'for' control-variable ':=' initial-value ( 'to' | 'downto' )
final-value
              'do' statement

  formal-parameter-list = '(' formal-parameter-section { ';' formal-
parameter-section } ')'
```

```
formal-parameter-section = value-parameter-specification |
            variable-parameter-specification |
            procedure-parameter-specification |
            function-parameter-specification

forward-directive = 'forward'

fractional-part = digit-sequence

function-block = block

function-declaration =
    function-heading ';' directive |
    function-identification ';' function-block |
    function-heading ';' function-block

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

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

function-identification = 'function' function-identifier

function-identifier = identifier

function-method-designator = object-variable '.' function-method-identifier
[ actual-parameter-list ]

function-method-identifier = identifier

function-parameter-specification = function-heading

goto-statement = 'goto' label

hex-digit = digit | 'a' | 'b' | 'c' | 'd' | 'e' | 'f'

hex-digit-sequence = hex-digit { hex-digit }

hexadecimal-integer = '$' hex-digit-sequence

identified-variable = pointer-variable '^' | pointer-variable '@'

identifier = letter { letter | digit }

identifier-list = identifier { ',' identifier }

if-statement = 'if' boolean-expression 'then' statement [ else-part ]

index-expression = expression

index-type = ordinal-type

index-type-list = index-type { ',' index-type }
```

```
indexed-variable = indexed-variable-array | indexed-variable-string

indexed-variable-array = array-variable '[' index-expression { ',' index-
expression } ']'

indexed-variable-string = string-variable '[' integral-expression ']'

initial-value = ordinal-expression

integer-number = decimal-integer | hexadecimal-integer | binary-integer

integral-constant = constant

integral-expression = expression

label = digit-sequence | identifier

label-declaration-group = 'label' label { ',' label } ';'

letter = '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' |
         '_'

list-type = 'list' 'of' component-type

list-variable = variable-access

max-string-length = '[' integral-constant ']' | '(' integral-constant ')'

member-designator = ordinal-expression [ '..' ordinal-expression ]

multiplying-operator = '*' | '/' | 'div' | 'mod' | 'and' | 'and_then'

name = string-literal

new-ordinal-type = enumerated-type | subrange-type

new-pointer-type = '^' domain-type | '@' domain-type

new-structured-type =
   [ 'packed' ] array-type |
   [ 'packed' ] record-type |
   [ 'packed' ] set-type |
   [ 'packed' ] file-type |
   [ 'packed' ] list-type |
                object-type |
                string-type

new-type = new-ordinal-type | new-structured-type | new-pointer-type

non-empty-string =
   ''' string-element-one string-element-one { string-element-one } ''' |
   '"' string-element-two string-element-two { string-element-two } '"'
```

object-type = 'object' | 'class'

object-variable = variable-access

ordinal-constant = constant

ordinal-expression = expression

ordinal-type = new-ordinal-type | ordinal-type-identifier

ordinal-type-identifier = identifier

pascal-string-type = 'string' [max-string-length]

pointer-variable = variable-access

printable-character = any character (including a space) that has a visual representation.

procedure-block = block

procedure-declaration =
    procedure-heading ';' directive |
    procedure-identification ';' procedure-block |
    procedure-heading ';' procedure-block

procedure-heading = 'procedure' identifier [formal-parameter-list]

procedure-identification = 'procedure' procedure-identifier

procedure-identifier = identifier

procedure-method-identifier = identifier

procedure-method-specifier = object-variable '.' procedure-method-identifier

procedure-method-statement = procedure-method-specifier [actual-parameter-list]

procedure-parameter-specification = procedure-heading

procedure-statement = procedure-identifier (
      [actual-parameter-list] |
      read-parameter-list | readln-parameter-list |
      write-parameter-list | writeln-parameter-list
)

program = program-heading ';' program-block

program-block = block

program-heading = 'program' identifier ['(' program-parameter-list ')']

```
program-parameter-list = identifier-list

property-designator = object-variable '.' property-specifier

property-identifier = identifier

property-specifier = property-identifier | '(' property-string ')'

property-string = string-expression

read-parameter-list = '(' [ file-variable ',' ] variable-access { ',' variable-
access } ')'

readln-parameter-list = [ '(' ( file-variable | variable-access ) { ',' variable-
access } ')' ]

real-number =
    digit-sequence '.' fractional-part [ exponent scale-factor ] |
    digit-sequence exponent scale-factor

record-section = identifier-list ':' type-denoter

record-type = 'record' field-list 'end'

record-variable = variable-access

record-variable-list = record-variable { ';' record-variable }

relational-operator = '=' | '<>' | '<' | '<=' | '>' | '>=' | 'in'

repeat-statement = 'repeat' statement-sequence 'until' boolean-
expression

repetitive-statement = repeat-statement | while-statement | for-statement

result-type = type-identifier

scale-factor = [ sign ] digit-sequence

selected-variable = list-variable '[' index-expression { ',' index-
expression } ']'

set-constructor = '[' [ member-designator { ',' member-designator } ] ']'

set-type = 'set' 'of' base-type

shift-expression = simple-expression [ shift-operator simple-expression ]

shift-operator = 'shl' | 'shr'

sign = '-' | '+'

simple-expression = term { adding-operator term }
```

```
simple-statement = empty-statement | assignment-statement |
        procedure-statement | procedure-method-statement |
        goto-statement

statement = [ label ':' ] ( simple-statement | structured-statement )

statement-part = compound-statement

statement-sequence = statement { ';' statement }

string-element-one = '''' | printable-character

string-element-two = '"""' | printable-character

string-expression = expression

string-literal = empty-string | non-empty-string

string-type = pascal-string-type | c-string-type

string-variable = variable-access

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

subrange-type = constant '..' constant

tag-field = identifier

tag-type = ordinal-type-identifier

term = factor { multiplying-operator factor }

type-definition = identifier '=' type-denoter

type-definition-group = 'type' type-definition ';' { type-definition ';' }

type-denoter = type-identifier | new-type

type-identifier = identifier

unsigned-constant = integer-number | real-number |
    character-literal | string-literal | constant-identifier |
    'nil'

value-parameter-specification = identifier-list ':' type-identifier

variable-access = entire-variable | component-variable | identified-variable
                                                                    |
        selected-variable | buffer-variable
```

variable-declaration = identifier-list ':' type-denoter

variable-declaration-group = 'var' variable-declaration { ';' variable-declaration }

variable-identifier = identifier

variable-parameter-specification = 'var' identifier-list ':' type-identifier

variant = case-constant-list ':' '(' field-list ')'

variant-body = variant-list [[';'] variant-part-completer] | variant-part-completer

variant-list = variant { ';' variant }

variant-part = 'case' variant-selector 'of' variant-body

variant-part-completer = ('otherwise' | 'else') (field-list)

variant-selector = [tag-field ':'] tag-type

while-statement = 'while' boolean-expression 'do' statement

with-statement = 'with' record-variable-list 'do' statement

write-parameter = expression [':' expression [':' expression]]

write-parameter-list = '(' [file-variable ','] write-parameter { ',' write-parameter } ')'

writeln-parameter-list = ['(' (file-variable | write-parameter) { ',' write-parameter } ')']