Zodiac Lexical Analysis

Index

[1 Lexical tokens 4](#_Toc432962147)

[1.1 General 4](#_Toc432962148)

[1.2 Special-symbols 4](#_Toc432962149)

[1.3 Identifiers 4](#_Toc432962150)

[1.4 Numbers 4](#_Toc432962151)

[1.5 char 5](#_Toc432962152)

[1.6 string 5](#_Toc432962153)

[1.7 Comment 5](#_Toc432962154)

[2 Type declarations and definitions 5](#_Toc432962155)

[2.1 Required-types 5](#_Toc432962156)

[2.1.1 General 5](#_Toc432962157)

[2.1.2 Simple-types 5](#_Toc432962158)

[2.1.3 List-type 5](#_Toc432962159)

[2.1.4 String-type 6](#_Toc432962160)

[2.2 Type-declarations 6](#_Toc432962161)

[2.3 Type-definitions 6](#_Toc432962162)

[2.3.1 General 6](#_Toc432962163)

[2.3.2 Structed-types 6](#_Toc432962164)

[3 Constructors 6](#_Toc432962165)

[3.1 General 6](#_Toc432962166)

[3.2 Default-constructors 6](#_Toc432962167)

[3.2.1 General 6](#_Toc432962168)

[3.2.2 Required-type-default-constructors 7](#_Toc432962169)

[3.2.3 Structed-type-default-constructors 7](#_Toc432962170)

[3.3 Defined-constructors 7](#_Toc432962171)

[3.3.1 General 7](#_Toc432962172)

[3.3.2 Required-type-defined-constructors 7](#_Toc432962173)

[4 Variables definitions and denoters 8](#_Toc432962174)

[4.1 Variable-definitions 8](#_Toc432962175)

[4.2 Entire-variables 8](#_Toc432962176)

[4.3 component-variables 8](#_Toc432962177)

[4.3.1 General 8](#_Toc432962178)

[4.3.2 Indexed-variables 8](#_Toc432962179)

[4.3.3 Field-designators 8](#_Toc432962180)

[4.4 Constructor-variables 9](#_Toc432962181)

[5 Function declarations and definition 9](#_Toc432962182)

[5.1 Function-declarations 9](#_Toc432962183)

[5.2 Function-definition 9](#_Toc432962184)

[5.2.1 General 9](#_Toc432962185)

[5.3 Function-blocks 9](#_Toc432962186)

[5.3.1 Function-parameter-blocks 9](#_Toc432962187)

[5.3.2 Function-instruction-blocks 9](#_Toc432962188)

[6 Expression 9](#_Toc432962189)

[6.1 General 9](#_Toc432962190)

[6.2 Operators 10](#_Toc432962191)

[6.3 Constant 10](#_Toc432962192)

[6.3.1 Gerneral 10](#_Toc432962193)

[6.3.2 Number-constants 10](#_Toc432962194)

[6.3.3 String-constants 10](#_Toc432962195)

[6.3.4 Boolean-constants 11](#_Toc432962196)

[6.3.5 Null-constants 11](#_Toc432962197)

[6.3.6 List-constants 11](#_Toc432962198)

[6.4 function-designator 11](#_Toc432962199)

[7 Statements 11](#_Toc432962200)

[7.1 General 11](#_Toc432962201)

[7.3 Simple-statements 11](#_Toc432962202)

[7.3.1 General 11](#_Toc432962203)

[7.3.2 Assignments-statements 11](#_Toc432962204)

[7.4 Structured-statements 12](#_Toc432962205)

[7.4.1 General 12](#_Toc432962206)

[7.4.2 Conditional-statements 12](#_Toc432962207)

[7.4.3 If-statements 12](#_Toc432962208)

[7.4.4 Repetitive-statements 12](#_Toc432962209)

[7.4.5 While-statements 12](#_Toc432962210)

[7.4.6 For-statements 12](#_Toc432962211)

[8 Scope 13](#_Toc432962212)

[8.1 General 13](#_Toc432962213)

[8.2 Scope-bodys 13](#_Toc432962214)

[8.3 Returns and escapes 13](#_Toc432962215)

[8.3.1 Returns 13](#_Toc432962216)

[8.3.2 Escapes 13](#_Toc432962217)

[9 Input and output 13](#_Toc432962218)

[9.1 General 13](#_Toc432962219)

[9.2 IO-constructors 13](#_Toc432962220)

[9.2.1 General 13](#_Toc432962221)

[9.2.2 IO-default-constructors 14](#_Toc432962222)

[9.2.3 IO-defined-constructors 14](#_Toc432962223)

[9.3 IO-reads 14](#_Toc432962224)

[9.4 IO-writes 14](#_Toc432962225)

[10 Programs 14](#_Toc432962226)

[10.1 General 14](#_Toc432962227)

[10.2 Program-headings 14](#_Toc432962228)

# 1 Lexical tokens

## 1.1 General

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' | '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'

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

## 1.2 Special-symbols

special-symbol ::= '(' | ') ' | ' [' | '] ' | '. ' | '! ' | '~' | '-' | '+' | '^^' | '\*' | '/' | '%' | '<<' | '>>' | '<' | '>' | '<=' | '>=' | '==' | '!= ' | '&' | '^' | '|' | '&&' | '||' | '=' | ':= ' | ''' | '"' | '@@' | '; ' | ', ' | '<-'

word-symbol ::= 'long' | 'real' | 'char' | 'bool' | 'list' | 'func' | 'oper' | 'var' | 'param' | 'return' | 'escape' | 'type' | 'family' | 'static' | 'True' | 'False' | 'if' | 'else' | 'for' | 'in' | 'while' | 'break' | 'continue' | 'from' | 'where' | 'select' | 'ance' | 'dsec' | 'IO' | 'have' | 'Null'

## 1.3 Identifiers

identifier ::= letter { letter | digit | '\_' }

## 1.4 Numbers

number ::= integer | real

integer ::= digit-sequence

real ::= digit-sequence '.' fractional-part

fractional-part ::= digit-sequence

digit-sequence ::= digit { digit }

## 1.5 char

char ::= ''' character '''

character ::= any ASCII unit

## 1.6 string

string ::= '[' string-content ']'

string-content ::= single-quotation-string | double-quotation-string

single-quotation-string ::= ''' { (character except ''' ) } '''

double-quotation-string ::= '"' { (character except '"' ) } '"'

## 1.7 Comment

comment ::= multi-line-comment | single-line-comment

multi-line-comment ::= '@{' multi-line-comment-content '@}'

single-line-comment ::= '@@' single-line-comment-content '\n'

multi-line-comment-content ::= { character except '@}' }

single-line-comment-content {character except '\n'}

# 2 Type declarations and definitions

## 2.1 Required-types

### 2.1.1 General

required-type ::= simple-type | list-type

### 2.1.2 Simple-types

simple-type ::= 'long' | 'real' | 'bool' | 'char'

### 2.1.3 List-type

list-type ::= 'list'

### 2.1.4 Func-type

func-type ::= 'func'

## 2.2 Type-declarations

type-declaration ::= 'type' type-identifier ';'

type-identifier ::= identifier

## 2.3 Type-definitions

### 2.3.1 General

type-definition ::= type-identifier [ '<-' type-identifier-list ] '{' structed-type '}'

type-identifier-list ::= type-identifier { ',' type-identifier }

type-identifier ::= identifier

### 2.3.2 Structed-types

structed-type ::= type-member { type-member }

type-member ::= member-variable | member-function

member-variable ::= 'var' variable-definition ';'

member-function ::= [ family ] function-definition

# 3 Constructors and Convertor

## 3.1 General

constructor ::= constructor | default-constructor

## 3.2 Default-constructors

### 3.2.1 General

default-constructor ::= required-type-default-constructor | structed-type-default-constructor

### 3.2.2 Required-type-default-constructors

required-type-default-constructor ::= required-type

### 3.2.3 Structed-type-default-constructors

structed-type-default-constructor ::= structed-type-identifier

structed-type-identifier ::= identifier

## 3.3 Defined-constructors

### 3.3.1 General

defined-constructor ::= required-type-defined-constructor | structed-type-defined-constructor

### 3.3.2 Required-type-defined-constructors

#### 3.3.2.1 General

required-type-defined-constructor ::= simple-type-defined-constructor | list-type-defined-constructor

#### 3.3.2.2 Simple-type-defined-constructors

simple-type-defined-constructor ::= simple-type '(' expression ')'

#### 3.3.2.3 List-type-defined-constructors

list-type-defined-constructor ::= list-type ( range-parameter | indexed-parameter )

range-parameter ::= '(' left-range ',' right-range ')'

left-range ::= expression

right-range ::= expression

indexed-parameter ::= '[' left-index , right-index ']' ( range-parameter | fill-parameter ).

fill-parameter ::= '(' expression ')'

left-index ::= expression

right-index ::= expression

## 3.4 Converter

### 3.4.1 General

converter ::= constructor | inherit-converters.

### 3.4.2 Inherit-converters

Inherit-converter ::= ansc-converter | dsec-converter

ansc-converter ::= 'ansc' '(' variable-identifier ')'

desc-converter ::= 'dsec' '(' variable-identifier ')'

# 4 Variables definitions

## 4.1 Variable-definitions

variable-default-definition ::= [ 'static' ] variable-identifier [ ',' variable-default-definition ',' | '=' ] default-constructor

variable-definition ::= [ 'static' ] variable-identifier [ ',' variable-definition ',' | '=' | ':=' ] variable-access

variable-access ::= entire-variable | component-variable | constructor-variable | converter-variable.

variable-identifier ::= identifier

## 4.2 Entire-variables

entire-variable ::= identifier

## 4.3 component-variables

### 4.3.1 General

component-variable ::= indexed-variable | field-designator

### 4.3.2 Indexed-variables

indexed-variable ::= array-variable '[' index-expression ']'

array-variable ::= variable-access

index-expression ::= expression

### 4.3.3 Field-designators

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

record-variable ::= variable-access

field-specifier ::= field-identifier

### 4.4 Constructor-variables

constructor-variable ::= constructor

### 4.5 Converter-variables

converter-variable ::= converter

# 5 Function declarations and definition

## 5.1 Function-declarations

function-declaration ::= [ 'static' ] 'func' function-identifier ';'

function-identifier ::= identifier

## 5.2 Function-definition

### 5.2.1 General

function-definition ::= [ 'static' ] 'func' function-identifier '{' function-body '}'

function-body ::= function-parameter-block function-instruction-block

## 5.3 Function-blocks

### 5.3.1 Function-parameter-blocks

function-parameter-block ::= { function-parameter }

function-parameter ::= 'param' variable-default-definition ';'

### 5.3.2 Function-instruction-blocks

function-instruction-block ::= [ function-scope-body ]

function-scope-body ::= scope-body

# 6 Operator overload

## 6.1 Gerneral

Operatortor-overload ::= 'oper' operator '{' function-body '}'

## 6.2 Operators

operator ::= self-operator | pow-operator | multiply-operator | add-operator | shift-operator | compare-operator | equal-operator | bit-and-operator | bit-xor-operator | bit-or-operator | and-operator | or-operator

self-operator ::= '!' | '~' | '+' | '-'

pow-operator ::= '^^'

multiply-operator ::= '\*' | '/' | '%'

add-operator ::= '+' | '-'

shift-operator ::= '<<' | '>>'

compare-operator ::= '<' | '>' | '>=' | '<='

equal-operator ::= '==' | '!='

bit-and-operator ::= '&'

bit-xor-operator ::= '^'

bit-or-operator ::= '|'

and-operator ::= '&&'

or-operator ::= '||'

# 7 Expression

## 7.1 General

expression ::= or-expression { or-operator or-expression }

or-expression ::= and-expression { and-operator and-expression }

and-expression ::= bit-or-expression { bit-or-operator bit-or-expression }

bit-or-expression ::= bit-xor-expression { bit-xor-operator bit-xor-expression }

bit-xor-expression ::= bit-and-expression { bit-and-operator bit-and-expression }

bit-and-expression ::= equal-expression { equal-operator equal-expression }

equal-expression ::= compare-expression { compare-operator compare-expression }

compare-expression ::= shift-expression { shift-operator shift-expression }

shift-expression ::= add-expression { add-operator add-expression }

add-expression ::= multiply-expression { multiply-operator multiply-expression }

multiply-expression ::= { pow-expressionpow-operator } pow-expression

pow-expression ::= [ self-operator ] term

term ::= variable-access | '(' expression ')' | function-designator

## 7.2 Constant

### 7.2.1 Gerneral

cosntant ::= number-constant | string-constant | boolean-constant | list-constant

### 7.2.2 Number-constants

number-constant ::= number

### 7.2.3 String-constants

string-constant ::= string

### 7.2.4 Boolean-constants

boolean-constant ::= 'True' | 'False'

7.2.5 Null-constants

null-constant ::= 'Null'

7.2.6 List-constants

list-constant ::= '[' list-expression-constant | list-select-constant ']'

list-expression-constant ::= expression { ',' expression }

list-select-constant ::= 'from' variable-identifier 'in' list 'where' boolean-expression 'select' variable-identifier

## 7.3 function-designator

function-designator ::= function-identifier '(' [ actual-parameter-list ] ')'

actual-parameter-list ::= actual-parameter { ',' actual-parameter }

actual-parameter ::= expression

# 8 Statements

## 8.1 General

statement ::= simple-statement | structed-statement | loop-statement | scope

## 8.2 Simple-statements

### 8.2.1 General

simple-statement ::= empty-statement | assignment-statement

empty-statement ::=

### 8.2.2 Assignments-statements

assignment-statement ::= variable-identifier [ ',' assignment-statement ',' | '=' | ':=' ] value

value ::= string | ''' character ''' | expression

## 8.3 Structured-statements

### 8.3.1 General

structured-statement ::= conditional-statement | if-statement | repetitive-statement | while-statement | for-statement

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

### 8.3.2 Conditional-statements

conditional-statement ::= if-statement

### 8.3.3 If-statements

if-statement ::= 'if' '(' boolean-expression ')' if-scope [ else-part ]

else-part ::= 'else' if-scope

if-scope ::= scope

boolean-expression ::= expression

### 8.3.4 Repetitive-statements

#### 8.3.4.1 General

repetitive-statement ::= while-statement | for-statement

#### 8.3.4.2 While-statements

while-statement ::= 'while' '(' boolean-expression ')' while-scope

while-scope ::= scope

#### 8.3.4.3 For-statements

for-statement ::= 'for' variable-identifier 'in' list for-scope

for-scope ::= scope

list ::= list-constructor | list-constant

list-constructor ::= list-type-defined-constructor | list-type-defined-constructor

## 8.4 Loop-statements

### 8.4.1 General

loop-statement ::= continue-statement | break-statement

### 8.4.2 Continue-statements

continue-statement ::= 'cotinue' '; '

### 8.4.3 Break-statements

break-statement ::= 'break' ';'

## 8.5 Ret-statements

### 8.5.1 General

ret-statement ::= return-statement | escape-statement

### 8.5.2 Return-statements

return-statement ::= ' return' [ actual-parameter-list ] '; '

### 8.5.3 Escape-statements

escape -statement ::= 'escape' actual-parameter-list ';'

# 9 Scope

## 9.1 General

scope ::= '{' { scope-body } '}'

## 9.2 Scope-bodys

scope-body ::= declaration | definition | statement | comment

declaration ::= function-declaration | type-declaration

definition ::= function-definition | type-definition | type-definition

# 10 Input and output

## 10.1 General

IO ::= IO-identifier | IO-constructor

IO-identifier ::= identifier

## 10.2 IO-constructors

### 10.2.1 General

IO-constructors ::= IO-default-constructor | IO-defined-constructor

### 10.2.2 IO-default-constructors

IO-default-constructor ::= IO-type

IO-type ::= 'IO'

### 10.2.3 IO-defined-constructors

IO-defined-constructor ::= IO-type '(' file-string, IO-mode-string ')'

file-string ::= string

IO-mode-string ::= string

## 10.3 IO-reads

IO-read ::= IO '.' read { '.' read }

read ::= 'read' '(' [actual-parameter] ')'

## 10.4 IO-writes

IO-write ::= IO '.' write { '.' write }

write ::= 'write' '(' [actual-parameter] ')'

# 11 Programs

## 11.1 General

program ::= program-heading | program-scope-body

program-scope-body ::= scope-body

## 11.2 Program-headings

program-heading-list ::= { have-sequence }

have-sequence ::= 'have' file-identifier '.d' ';'

file-identifier ::= identifier