Heine-fluch EBNF Grammar

Program

program = block “.” ;

block = declaration_part statement_part ;

Declaration Part

declaration_part = {(label_declaration_part | constant_declaration_part |
variable_declaration_part | procedure_declaration_part)} ;

Label Declaration

label_declaration_part = “label” label { “,” label } “;” ;

Constant Declaration

constant_declaration_part = “const” type constant_declaration “;” { constant_declaration “;” } ;

constant_declaration = identifier_list “:=” constant ;

Variable Declaration

variable_declaration_part = variable_simple_declaration “;” | variable_paralel_declaration “;” ;

variable_simple_declaration = type identifier_list “:=” expression_list | type identifier_list ;

variable_paralel_declaration = type identifier_list “:=” “[“ expression_list “]” ;

Procedure Declaration

procedure_declaration_part = procedure_heading “;” procedure_body ;

procedure_body = block ;

procedure_heading = “procedure” identifier ;

Statement Part

statement_part = “begin” statement_sequence “end” ;

statement_sequence = statement { “;” statement } ;

statement = label “:” (simple_statement | structured_statement) | (simple_statement |
structured_statement) ;

Simple Statement

simple_statement = (assignment_statement | procedure_statement | goto_statement |
ternary_statement) ;

Ternary Statement

ternary_statement = identifier “:=” expression “?” expression “:” expression ;

Assignment Statement
assignment_statement = identifier “:=” expression ;

Goto Statement
goto_statement = “goto” label ;

Procedure Statement
procedure_statement = “call” identifier ;

Structured Statement
structured_statement = (compound_statement | repetitive_statement | conditional_statement) ;

Compound Statement
compound_statement = “begin” statement_sequence “end” ;

Repetitive Statement
repetetive_statement = (while_do_statement | do_while_statement | repeat_statement | for_statement) ;

While Do Statement
while_do_statement = “while” expression “do” statement ;

Do While Statement
do_while_statement = “do” statement “while” expression ;

Repeat Statement
repeat_statement = “repeat” statement_sequence “until” expression ;

For Statement
for_statement = “for” identifier “:=” expression (“to” | “downto”) expression “do” statement ;

Conditional Statement
conditional_statement = (if_statement | case_statement) ;

If Statement
if_statement = “if” expression “then” statement [“else” statement];

Case Statement
case_statement = “case” expression “of” case_limb { “;” case_limb } [“;”] “end” ;

case_limb = case_label_list “:” statement ;

Low Level Definitions
identifier = ('a' .. 'z' | 'A' .. 'Z') { 'a' .. 'z' | 'A' .. 'Z' | '0' .. '9' | '_' } ;

constant = [sign] (identifier | number) | string ;

type = (“string” | “real” | “integer” | “boolean”) ;

label = integer_number ;

identifier_list = identifier { “,” identifier } ;

```
expression_list = expression { “,” expression } ;

case_label_list = constant { “,” constant } ;

expression = simple_expression { relational_operator simple_expression } ;

simple_expression = [ sign ] term { addition_operator term } ;

term = factor { multiplication_operator factor } ;

factor = ( number | string | identifier | “(“ expression “)” ) ;

relational_operator = ( “=” | “<>” | “<” | “<=” | “>” | “>=” ) ;

addition_operator = ( “+” | “-“ | “or” ) ;

multiplication_operator = ( “*” | “/” | “and” )

string = “ ’ ” string_character { string_character } “ ‘ “ ;

string_character = any-character-except-quote | “ ‘ “ ;

number = ( integer_number | real_number ) ;

integer_number = digit_sequence ;

real_number = digit_sequence “.” { unsigned_digit_sequence } | digit_sequence ;

digit_sequence = [ sign ] unsigned_digit_sequence ;

unsigned_digit_sequence = digit { digit } ;

digit = '0' .. '9' ;

sign = ( “+” | “-“ ) ;
```