Para el caso de vacío(empty) se usa el character E en ambas formas

***BNF***

program ::= ‘program’ identifier ‘;’ block ’.’ ;

block ::= Variable.declaration.part

Procedure.declaration.part

Statement.part ;

Variable.declaration.part ::= E | ‘var’ variable.declaration ;

variable.declaration ::= var.dec ‘;’| variable.declaration var.dec ‘;’ ;

var.dec ::= identifier.part ‘:’ type ;

identifier.part ::= identifier | identifier.part ‘,’ identifier ;

type ::= Simple.type | Array.type;

Array.type ::= ‘array [’ index.range ‘] of’ Simple.type ;

index.range ::= integer.constant ‘..’ integer.constant ;

Simple.type ::= identifier ;

Procedure.declaration.part ::= E | Procedure.declaration.part

Procedure.declaration ;

Procedure.declaration ::= ‘Procedure’ identifier ‘;’ block ;

Statement.part ::= ‘begin’ list.statement ‘end’ ;

list.statement ::= statement | list.statement ‘;’ statement ;

statement ::= Simple.statement | Structured.statement ;

Simple.statement ::= assignament.statement | procedure.statement |

read.statement | write.statement ;

assignament.statement ::= variable ‘::=’ expression ;

procedure.statement ::= identifier ;

read.statement ::= ‘read (’ list.variable ‘)’ ;

list.variable ::= variable | list.variable ‘,’ variable ;

write.statement ::= ‘write (’ list.expression ‘)’ ;

list.expression ::= expression | list.expression ‘,’ expression ;

Structured.statement ::= Statement.part | if.statement |

while.statement ;

if.statement ::= ‘if’ expression ‘then’ statement | ‘if’ expression

‘then’ statement ‘else’ statement ;

while.statement ::= ‘while’ expression ‘do’ statement ;

expression ::= Simple.expression | Simple.expression relational.operator

Simple.expression ;

Simple.expression ::= sign term list.addope ;

list.addope ::= adding.operator term | E | list.addope adding.operator term ;

term ::= factor list.multiope ;

list.multiope ::= E | multiplying.operator factor | list.multiop

multiplying.operator factor;

factor ::= variable | constant | ‘(’ expression ‘)’ | ‘not’ factor ;

relational.operator ::= ‘=’ | ‘<>’ | ‘<’ | ‘<=’ | ‘>=’ | ‘>’ ;

sign ::= ‘+’ | ‘-‘ | E ;

adding.operator ::= ‘+’ | ‘-‘ | ‘or’ ;

multiplying.operator ::= ‘\*’ | ‘div’ | ‘and’ ;

variable ::= identifier | indexed.variable ;

indexed.variable ::= identifier ‘[’ expression ‘]’ ;

constant ::= integer.constant | character.constant | identifier ;

identifier ::= letter | identifier letter.or.digit ;

letter.or.digit ::= letter | digit ;

integer.constant ::= digit | integer.constant digit ;

character.constant ::= ‘***‘***’ any character other than *’* ‘*’*’ | **‘***‘‘’’***’** ;

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’ ;

Special.symbol ::= ‘+’ | ‘-‘ | ‘\*’ | ‘=’ | ‘<>’ | ‘<’ | ‘>’ | ‘<=’ |

‘>=’ | ‘(‘ | ‘)’ | ‘[‘ | ‘]’ | ‘:=’ | ‘.’ | ‘,’ |

‘;’ | ‘:’ | ‘..’ | ‘div’ | ‘or’ | ‘and’ | ‘not’ | ‘if’ |

‘then’ | ‘else’ | ‘of’ | ‘while’ | ‘do’ | ‘begin’ | ‘end’ |

‘read’ | ‘write’ | ‘var’ | ‘array’ | ‘procedure’ |

‘program’ ;

predefined.identifier ::= ‘Integer’ | ‘Boolean’ | ‘true’ | ‘false’ ;

***EBNF***

program ::= ‘program’ identifier ‘;’ block ’.’ ;

block ::= Variable.declaration.part

Procedure.declaration.part

Statement.part ;

Variable.declaration.part ::= (‘var’ (variable.declaration ‘;’)+)? ;

variable.declaration ::= identifier (‘;’ identifier)+ ‘:’ type ;

type ::= Simple.type | Array.type;

Array.type ::= ‘array [’ index.range ‘] of’ Simple.type ;

index.range ::= integer.constant ‘..’ integer.constant ;

Simple.type ::= identifier ? ;

Procedure.declaration.part ::= (Procedure.declaration ‘;’)\* ;

Procedure.declaration ::= ‘Procedure’ identifier ‘;’ block ;

Statement.part ::= ‘begin’ statement(‘;’ statement)+ ‘end’ ;

statement ::= Simple.statement | Structured.statement ;

Simple.statement ::= assignament.statement | procedure.statement |

read.statement | write.statement ;

assignament.statement ::= variable ‘::=’ expression ;

procedure.statement ::= identifier ;

read.statement ::= ‘read (’ variable ( ‘;’ variable)\* ‘)’ ;

write.statement ::= ‘write (’ expression (‘;’ expression)\* ‘)’;

Structured.statement ::= Statement.part | if.statement |

while.statement ;

if.statement ::= ‘if’ expression ‘then’ statement | ‘if’ expression

‘then’ statement ‘else’ statement ;

while.statement ::= ‘while’ expression ‘do’ statement ;

expression ::= Simple.expression | simple.expression relational.operator

Simple.expression ;

Simple.expression ::= sign term (adding.operator term)\* ;

term ::= factor(multiplying.operator factor)\* ;

factor ::= variable | constant | ‘(’ expression ‘)’ | ‘not’ factor ;

relational.operator ::= ‘=’ | ‘<>’ | ‘<’ | ‘<=’ | ‘>=’ | ‘>’ ;

sign ::= ‘+’ | ‘-‘ | E ;

adding.operator ::= ‘+’ | ‘-‘ | ‘or’ ;

multiplying.operator ::= ‘\*’ | ‘div’ | ‘and’ ;

variable ::= identifier | indexed.variable ;

indexed.variable ::= identifier ‘[’ expression ‘]’ ;

constant ::= identifier.constant | character.constant | identifier ;

identifier ::= [a-zA-Z] | [a-zA-Z0-9]\* ;

integer.constant ::= [0-9]+ ;

character.constant ::= ‘***‘***’ any character other than *’* ‘*’*’ | **‘***‘‘’’***’** ;

Special.symbol ::= ‘+’ | ‘-‘ | ‘\*’ | ‘=’ | ‘<>’ | ‘<’ | ‘>’ | ‘<=’ |

‘>=’ | ‘(‘ | ‘)’ | ‘[‘ | ‘]’ | ‘:=’ | ‘.’ | ‘,’ |

‘;’ | ‘:’ | ‘..’ | ‘div’ | ‘or’ | ‘and’ | ‘not’ |

‘if’ | ‘then’ | ‘else’ | ‘of’ | ‘while’ | ‘do’ |

‘begin’ | ‘end’ | ‘read’ | ‘write’ | ‘var’ | ‘array’ |

‘procedure’ |‘program’ ;

predefined.identifier ::= ‘Integer’ | ‘Boolean’ | ‘true’ | ‘false’ ;