Pasc500

A. TOKENS

Note that Pasc500 is case-insensitive so there isn't distinction between uppercase and lowercase alphabetic characters unless they are part of a CCONST word or STRING.

KEYWORDS

```
PROGRAM, CONST, TYPE, ARRAY, SET, OF, RECORD, VAR, FORWARD,
FUNCTION, PROCEDURE, INTEGER, REAL, BOOLEAN, CHAR, BEGIN, END,
IF, THEN, ELSE, WHILE, DO, FOR, DOWNTO, TO, WITH, READ, WRITE
```

ID

Strings that begin with an optional '_' character, followed by an alphabetic character, followed by zero or more alphanumeric characters or '_' characters, and are not keywords.

Accepted examples:

```
a100__version_2
_a100__version2
```

Unaccepted examples:

```
100__version_2
a100__version2_
_100__version_2
a100--version-2
```

ICONST

The unique character '0', which represents the constant with a value of 0. Also, one or more numeric characters, the first of which is not '0', in which case the value represented is the corresponding number in decimal. Also, the string "0H" followed by one or more numeric characters the first of which is not '0', or by one or more of the alphabetic characters 'A', 'B', 'C', 'D', ' E' and 'F'. In this case, the value represented is the corresponding number – after the prefix "0H" – in hexadecimal. Finally, the string "0B" followed by one or more of the numeric characters '0' and '1', the first of which is not '0', so the value represented is the corresponding number in binary.

Accepted examples:

```
0
180
0H9F0
0B1001
```

Unaccepted examples:

```
0180
HB7
0H0
0B010
0HG8A
```

RCONST

Zero or more numeric characters followed by the character "." and at least one numeric character. An optional exponent field follows, beginning with the character 'E', followed by an optional sign and at least one numeric character.

Alternatively, unsigned integer constant that must be followed by an exponent field. If there is no exponent field, the number can be in hexadecimal, prefixed with “0H”, or in binary, prefixed with “0B”. The integer part of a real constant, like the numeric part of the exponent, cannot start with '0' if it is not '0'. Whenever there is a fractional part, it must contain at least one character other than '0', unless it is '0'.

Accepted examples:

```
180E-2
.5
180.100
7.0
0HA.9
0H.00B9CF
0B1.1001
```

Unaccepted examples:

```
180E-2.2
.E-2
180E
1100
5.
7.00
.5G-2
05.2E-05
0HBE-2
```

BCONST

The keywords “TRUE” and “FALSE”

CCONST

Any ASCII character (32-126) between two occurrences of the special character “ ‘ “ . Additional, special ASCII characters are represented with ‘\’ character. More specifically, the LF (Line Feed) character is represented as '\n', the FF (Form Feed) character as '\f', the HT (Horizontal Tab) character as '\t', the CR (Carriage Return) as '\r', the BS (BackSpace) character as '\b' and the VT (Vertical Tab) character as '\v'.

Accepted examples:

```
'a'
'$'
' '
'''
'\n'
'\'
```

Unaccepted examples:

```
'ac'
'\p'
'\\'
```

OPERATORS

Comparison operators (RELOP): `> >= < <= <>`
Addition and subtraction operators (ADDOP): `+ -`
Logic OR (OROP): `OR`
Multiplicative operators (MULDIVANDOP): `* / DIV MOD AND`
Logic NOT (NOTOP): `NOT`
Membership IN (INOP): `IN`

STRINGS

Any string between two occurrences of the special character ' “ '. The character ' “ ' and the above ASCII special characters are represented in a character string using the '\' character. Any other use of the '\' character represents the following character. Thus, the '\' character itself is represented as \\. Especially when the '\' character is at the end of the line, the string continues to the next line, without the '\' and newline characters being part of it.

Accepted examples:

```
"CHARACTER +"
"STRINGS START AND END WITH \""
"CHARACTER \\ AT THE END OF THE LINE \
EXTENDS STRING IN THE NEXT LINE\n"
```

OTHER TOKENS

‘(’ (LPAREN), ‘)’ (RPAREN), ‘;’ (SEMI), ‘.’ (DOT), ‘,’ (COMMA), ‘=’ (EQU), ‘:’ (COLON), ‘[’ (LBRACK), ‘]’ (RBRACK), “:=” (ASSIGN), “..” (DOTDOT), ‘<EOF>’ (EOF)

The EOF token does not appear in the PASC500 grammar, but must be produced by the parser with a value of 0 to terminate parsing.

COMMENTS

Comments in PASC500 are strings enclosed by the character pair ‘{’ , ‘}’.

B. SYNTAX RULES

```
program → header declarations subprograms comp_statement DOT
header → PROGRAM ID SEMI
declarations → constdefs typedefs vardefs
constdefs → CONST constant_defs SEMI
| ε
constant_defs → constant_defs SEMI ID EQU expression
| ID EQU expression
expression → expression RELOP expression
| expression EQU expression
| expression INOP expression
| expression OROP expression
| expression ADDOP expression
| expression MULDIVANDOP expression
| ADDOP expression
| NOTOP expression
| variable
| ID LPAREN expressions RPAREN
| constant
| LPAREN expression RPAREN
| setexpression
variable → ID
| variable DOT ID
| variable LBRACK expressions RBRACK
expressions → expressions COMMA expression
| expression
constant → ICONST
| RCONST
| BCONST
| CCONST
setexpression → LBRACK elexpressions RBRACK
| LBRACK RBRACK
elexpressions → elexpressions COMMA elexpression
| elexpression
elexpression → expression DOTDOT expression
| expression
typedefs → TYPE type_defs SEMI
| ε
type_defs → type_defs SEMI ID EQU type_def
| ID EQU type_def
type_def → ARRAY LBRACK dims RBRACK OF typename
| SET OF typename

| RECORD fields END
| LPAREN identifiers RPAREN
| limit DOTDOT limit
dims → dims COMMA limits
| limits
limits → limit DOTDOT limit
| ID
limit → ADDOP ICONST
| ADDOP ID
| ICONST
| CCONST
| BCONST
| ID
typename → standard_type
```

```
| ID
standard_type → INTEGER | REAL | BOOLEAN | CHAR
fields → fields SEMI field
| field
field → identifiers COLON typename
identifiers → identifiers COMMA ID
| ID
vardefs → VAR variable_defs SEMI
| ε
variable_defs → variable_defs SEMI identifiers COLON typename
| identifiers COLON typename
subprograms → subprograms subprogram SEMI
| ε
subprogram → sub_header SEMI FORWARD
| sub_header SEMI declarations subprograms comp_statement
sub_header → FUNCTION ID formal_parameters COLON standard_type
| PROCEDURE ID formal_parameters
| FUNCTION ID
formal_parameters → LPAREN parameter_list RPAREN
| ε
parameter_list → parameter_list SEMI pass identifiers COLON
typename
| pass identifiers COLON typename
pass → VAR | ε
comp_statement → BEGIN statements END
statements → statements SEMI statement
|statement

statement → assignment
| if_statement
| while_statement
| for_statement
| with_statement
| subprogram_call
| io_statement
| comp_statement
| ε
assignment → variable ASSIGN expression
| variable ASSIGN STRING
if_statement → IF expression THEN statement if_tail
if_tail → ELSE statement
| ε
while_statement → WHILE expression DO statement
for_statement → FOR ID ASSIGN iter_space DO statement
iter_space → expression TO expression
| expression DOWNTO expression
with_statement → WITH variable DO statement
subprogram_call → ID
| ID LPAREN expressions RPAREN
io_statement → READ LPAREN read_list RPAREN
| WRITE LPAREN write_list RPAREN
read_list → read_list COMMA read_item
| read_item
read_item → variable
write_list → write_list COMMA write_item
| write_item
write_item → expression
|STRING
```

where the symbol '|' separates the alternate right members of the rules and ε is the empty string.

The above rules define an ambiguous grammar, which with the operator precedence and associativity can be made unambiguous.

The initial start symbol of PASC500 is "program".

Operator	Precedence	Associativity
'.' , '[]' , '()'	Highest	Left to right
NOTOP		-
MULDIVANDOP		Left to right
ADDOP , OROP		Left to right
INOP , RELOP , '='	Lowest	Left to right