CD END SEM

G4

```
grammar Spec;

@parser::header {import ast.*;}
program           : programHeader block '.'
                          ;
programHeader     : PROGRAM IDENTIFIER programParameters? ';'
                          ;

programParameters : '(' IDENTIFIER ( ',' IDENTIFIER )* ')' ;

block         : declarations compoundStatement ;
declarations  : ;

statement : compoundStatement
          | assignmentStatement
          | repeatStatement
          | writeStatement
          | writelnStatement
          | emptyStatement
          | whileStatement
          | forStatement
          | ifStatement
          | ifElseStatement
          | switchStatement
          ;

whileStatement : WHILE expression DO statement;

forStatement : FOR IDENTIFIER ':=' expression ( TO | DOWNTO ) expression
DO statement;

ifStatement : IF expression THEN statement;

ifElseStatement : IF expression THEN statement ELSE statement;

switchStatement : CASE expression OF caseStatement END
                        | CASE expression OF END ;

caseStatement : constantList ':' statement
                   | constantList ':' statement ';'  caseStatement;

constantList : constant
                  | constant ',' constantList
```

```
                  ;

constant : stringConstant
             | number
             | sign IDENTIFIER
             ;

compoundStatement : BEGIN statementList END ;
emptyStatement    : ;

statementList       : statement ( ';' statement )* ;

assignmentStatement : lhs ':=' rhs ;
repeatStatement     : REPEAT statementList UNTIL expression ;

lhs : variable ;
rhs : expression ;

writeStatement   : WRITE writeArgumentsOn ;
writelnStatement : WRITELN writeArgumentsLn? ;

expression
    : simpleExpression
    | simpleExpression relOp simpleExpression
    ;

simpleExpression
    : sign? term (addOp term)* ;

term
    : factor (mulOp factor)* ;

factor
    : variable              # variableExpression
    | number                # numberExpression
    | characterConstant     # characterFactor
    | stringConstant        # stringFactor
    | NOT factor            # notFactor
    | '(' expression ')'    # parenthesizedExpression
    ;

variable : IDENTIFIER ;

number          : sign? unsignedNumber ;
unsignedNumber  : integerConstant | realConstant ;
integerConstant : INTEGER ;
```

```
realConstant    : REAL;

characterConstant : CHARACTER ;
stringConstant    : STRING ;

sign : '-' | '+' ;

relOp : '=' | '<>' | '<' | '<=' | '>' | '>=' ;
addOp : '+' | '-' | OR ;
mulOp : '*' | '/' | DIV | MOD | AND ;

writeArgumentsOn    : '(' writeArgumentListOn ')' ;
writeArgumentListOn : writeArgumentList ;
writeArgumentsLn    : '(' writeArgumentListLn ')' ;
writeArgumentListLn : writeArgumentList ;
writeArgumentList   : writeArgument (',' writeArgument)* ;
writeArgument       : expression (':' fieldWidth)? ;
fieldWidth          : sign? integerConstant (':' decimalPlaces)? ;
decimalPlaces       : integerConstant ;

fragment A : ('a' | 'A') ;
fragment B : ('b' | 'B') ;
fragment C : ('c' | 'C') ;
fragment D : ('d' | 'D') ;
fragment E : ('e' | 'E') ;
fragment F : ('f' | 'F') ;
fragment G : ('g' | 'G') ;
fragment H : ('h' | 'H') ;
fragment I : ('i' | 'I') ;
fragment J : ('j' | 'J') ;
fragment K : ('k' | 'K') ;
fragment L : ('l' | 'L') ;
fragment M : ('m' | 'M') ;
fragment N : ('n' | 'N') ;
fragment O : ('o' | 'O') ;
fragment P : ('p' | 'P') ;
fragment Q : ('q' | 'Q') ;
fragment R : ('r' | 'R') ;
fragment S : ('s' | 'S') ;
fragment T : ('t' | 'T') ;
fragment U : ('u' | 'U') ;
fragment V : ('v' | 'V') ;
fragment W : ('w' | 'W') ;
fragment X : ('x' | 'X') ;
fragment Y : ('y' | 'Y') ;
fragment Z : ('z' | 'Z') ;
```

```
PROGRAM   : P R O G R A M ;
CONST     : C O N S T ;
TYPE      : T Y P E ;
ARRAY     : A R R A Y ;
OF        : O F ;
RECORD    : R E C O R D ;
VAR       : V A R ;
BEGIN     : B E G I N ;
END       : E N D ;
DIV       : D I V ;
MOD       : M O D ;
AND       : A N D ;
OR        : O R ;
NOT       : N O T ;
IF        : I F ;
THEN      : T H E N ;
ELSE      : E L S E ;
CASE      : C A S E ;
REPEAT    : R E P E A T ;
UNTIL     : U N T I L ;
WHILE     : W H I L E ;
DO        : D O ;
FOR       : F O R ;
TO        : T O ;
DOWNTO    : D O W N T O ;
WRITE     : W R I T E ;
WRITELN   : W R I T E L N ;
READ      : R E A D ;
READLN    : R E A D L N ;
PROCEDURE : P R O C E D U R E ;
FUNCTION  : F U N C T I O N ;

IDENTIFIER : [a-zA-Z][a-zA-Z0-9]* ;
INTEGER    : [0-9]+ ;

REAL       : INTEGER '.' INTEGER
           | INTEGER ('e' | 'E') ('+' | '-')? INTEGER
           | INTEGER '.' INTEGER ('e' | 'E') ('+' | '-')? INTEGER
           ;

NEWLINE : '\r'? '\n' -> skip  ;
WS      : [ \t]+ -> skip ;

QUOTE     : '\'' ;
CHARACTER : QUOTE CHARACTER_CHAR QUOTE ;
```

```
STRING     : QUOTE STRING_CHAR* QUOTE ;

fragment CHARACTER_CHAR : ~('\'')   // any non-quote character
                        ;

fragment STRING_CHAR : QUOTE QUOTE  // two consecutive quotes
                     | ~('\'')      // any non-quote character
                     ;
```

Parsing of testcase.txt