```
grammar SimpleExpr;

@header {
package simpleexpr;
}

// The name of a grammar/syntax rule starts with a lowercase letter.
// Each grammar rule has two parts: a head and a body, separated by a
 ':'
// *: 0 or more
// EOF: end of file
prog : stat* EOF ;

// | : or (choices)
// '=': literal
// 'if': also literal, in single quote
// literals are treated implicitly as lexer rules;
// put in between grammar rules and explict lexer rules.
stat : expr ';'         # ExprStat
     | ID '=' expr ';'  # AssignStat
     | 'if' expr ';'    # IfStat
     ;

// (): subrule
// vs. '(' ... ')'
expr : expr ('*' | '/') expr    # MulDivExpr
     | expr ('+' | '-') expr    # AddSubExpr
     | '(' expr ')'             # ParenExpr
     | ID                       # IdExpr
     | INT                      # IntExpr
     ;

// The name of a lexer rule starts with a uppercase letter.
// Usually, such a name consists of only uppercase letters.
// (): subrule
ID : (LETTER | '_') WORD* ;

INT : '0' | ([1-9] [0-9]*) ;

WS : [ \t\r\n]+ -> skip ;


// Note: in antlr4, '.' matches any (single) character, including '\n
'.

// can be used as Java constants (public static final int)
SEMI : ';' ;
EQUAL : '=' ;
IF : 'if' ;
MUL : '*' ;
DIV : '/' ;
ADD : '+' ;
SUB : '-' ;
```

```
LPAREN : '(' ;
RPAREN : ')' ;

// .: match any single character
// .*: match 0 or more characters
// *?: non-greedy
SL_COMMENT : '//' .*? '\n' -> skip;  // non-greedy

// ~: except
SL_COMMENT2 : '//' ~[\r\n]* '\r'? '\n' -> skip;  // non-greedy

ML_COMMENT : '/*' .*? '*/' -> skip;  // non-greedy

// These are helper token rules.
// They will not generate tokens.
fragment DIGIT : [0-9] ;
fragment LETTER : [a-zA-Z] ;
fragment WORD : [a-zA-Z0-9_] ;
```