Bai 1

grammar BKOOL;

@lexer*::header* {

from lexererr import \*

}

options {

    language = Python3;

}

*// Use ANTLR to write regular expressions describing Pascal strings are made up of a sequence of characters between single quotes: 'string'. The*

*// single quote itself can appear as two single quotes back to back in a string: 'isn''t'.*

program: (expr NEWLINE)\* EOF;

expr: expr ('\*' | '/') expr | expr ('+' | '-') expr | '(' expr ')';

NEWLINE: [\r\n]+;

ID: [a-z] [a-z0-9]\*;

*// REAL: ([0-9]+ '.' [0-9]+ ([eE] [+-]? [0-9]+)?) | ([0-9]+ [eE] [+-]? [0-9]+);*

*// STRING: '\'' ( ~('\'' | '\r' | '\n') | '\'\'')\* '\'';*

WS: [ \t\r\n]+ -> skip; *// skip spaces, tabs, newlines*

ERROR\_CHAR: . {raise ErrorToken(self.text)};

UNCLOSE\_STRING: .;

ILLEGAL\_ESCAPE: .;

Bai 2

grammar BKOOL;

@lexer*::header* {

from lexererr import \*

}

options {

    language = Python3;

}

*// Use ANTLR to write regular expressions describing Pascal strings are made up of a sequence of characters between single quotes: 'string'. The*

*// single quote itself can appear as two single quotes back to back in a string: 'isn''t'.*

program: (expr NEWLINE)\* EOF;

expr: expr ('\*' | '/') expr | expr ('+' | '-') expr | '(' expr ')';

NEWLINE: [\r\n]+;

*// ID: [a-z] [a-z0-9]\*;*

REAL: ([0-9]+ '.' [0-9]+ ([eE] [+-]? [0-9]+)?) | ([0-9]+ [eE] [+-]? [0-9]+);

*// STRING: '\'' ( ~('\'' | '\r' | '\n') | '\'\'')\* '\'';*

WS: [ \t\r\n]+ -> skip; *// skip spaces, tabs, newlines*

ERROR\_CHAR: . {raise ErrorToken(self.text)};

UNCLOSE\_STRING: .;

ILLEGAL\_ESCAPE: .;

Bai 3

grammar BKOOL;

@lexer*::header* {

from lexererr import \*

}

options {

    language = Python3;

}

*// Use ANTLR to write regular expressions describing Pascal strings are made up of a sequence of characters between single quotes: 'string'. The*

*// single quote itself can appear as two single quotes back to back in a string: 'isn''t'.*

program: (expr NEWLINE)\* EOF;

expr: expr ('\*' | '/') expr | expr ('+' | '-') expr | '(' expr ')';

NEWLINE: [\r\n]+;

*// ID: [a-z] [a-z0-9]\*;*

REAL: ([0-9]+ '.' [0-9]+ ([eE] [+-]? [0-9]+)?) | ([0-9]+ [eE] [+-]? [0-9]+);

STRING: '\'' ( ~('\'' | '\r' | '\n') | '\'\'')\* '\'';

WS: [ \t\r\n]+ -> skip; *// skip spaces, tabs, newlines*

ERROR\_CHAR: . {raise ErrorToken(self.text)};

UNCLOSE\_STRING: .;

ILLEGAL\_ESCAPE: .;s