parser grammar ExprParser;

options { tokenVocab=ExprLexer; }

program

: statementList

;

modifier : M\_FLOAT

| M\_BOOLEAN

| M\_INTEGER

| PARAM

| VECTOR '[' INT ']'

| TYPE '(' ID ')'

;

idName: ID;

idList : idName

| idName ',' idList

;

modifierList : modifier

| modifierList modifier

;

nameDecl : modifierList idList;

codeFragment : START\_FRAGMENT idName ';' literal END\_FRAGMENT

| START\_FRAGMENT idName ';' statementList END\_FRAGMENT

| START\_FRAGMENT modifierList idName ';' INT END\_FRAGMENT

| START\_FRAGMENT modifierList idName ';' statementList END\_FRAGMENT

| START\_FRAGMENT statementList END\_FRAGMENT

;

statementList : statement statementList

| statement

;

statement : nameDecl ';'

| destiny '=' expr ';'

| callFunc ';'

| codeFragment

| IF '(' expr ')' statementList ELSE statementList

| IF '(' expr ')' statementList

| WHILE '(' expr ')' statementList

| SELECT '(' expr ')' START\_FRAGMENT caseBlock END\_FRAGMENT

| PRINT '(' expressionList ')' ';'

| READ '(' expressionList ')' ';'

| BREAK ';'

;

destiny : idName

| idName '[' expr ']'

| name

;

caseBlock : CASE expr ':' statementList caseBlock

| DEFAULT ':' statementList caseBlock

;

expressionList : expr

| expressionList ',' expr

;

expr: primary

| unaryOp expr

| expr binOp expr

;

primary : name

| callFunc

| '(' expr ')'

| idName '(' expressionList ')'

| TRUE

| FALSE

| literal

;

literal : INT

| STRING

;

name : idName

| idName '[' expr ']'

| name '.' name

;

unaryOp : SUB

| NOT

| SUM

;

binOp : EQS

| GREATER

| GREATER\_EQ

| SMALLER

| SMALLER\_EQ

| DIFF

| SUM

| SUB

| MUL

| EQ

| DIV

| AND

| OR

;

callFunc : idName '(' expressionList ')' ;