grammar CG;

ADD : '+' ;

ADD\_ASSIGN : '+=' ;

AND : '&&' ;

AND\_ASSIGN : '&&=' ;

BOOL : 'bool' ;

BREAK : 'break' ;

COMMA : ',' ;

CONTINUE : 'continue' ;

DIGRAPH : 'digraph' ;

DIV : '/' ;

DIV\_ASSIGN : '/=' ;

ELIF : 'elif' ;

ELSE : 'else' ;

EQUAL : '=' ;

EQUAL\_EQUAL : '==' ;

FLOAT : 'float' ;

FOR : 'for' ;

FUNC : 'func' ;

GE : '>=' ;

GRAPH : 'graph' ;

GT : '>' ;

IF : 'if' ;

INT : 'int' ;

LBRACE : '{' ;

LE : '<=' ;

LPAREN : '(' ;

LT : '<' ;

ML\_COMMENT : '/\*' (.)\*? '\*/' -> skip ;

MOD : '%' ;

MOD\_ASSIGN : '%=' ;

MUL : '\*' ;

MUL\_ASSIGN : '\*=' ;

NOT : '!' ;

NOT\_EQUAL : '!=' ;

OR : '||' ;

OR\_ASSIGN : '||=' ;

RBRACE : '}' ;

RETURN : 'return' ;

RPAREN : ')' ;

SEMI : ';' ;

STRING : 'string' ;

SUB : '-' ;

SUB\_ASSIGN : '-=' ;

VAR : 'var' ;

VOID : 'void' ;

WHILE : 'while' ;

LBRACK : '[' ;

RBRACK : ']' ;

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

BoolLit: 'true' | 'false';

fragment Letter: [a-zA-Z\_];

fragment DecimalDigit: [0-9];

Identifier: Letter (Letter | DecimalDigit)\*;

fragment DecimalLit: DecimalDigit+;

IntLit: DecimalLit;

fragment Exponent: [eE] [+-]? DecimalLit;

FloatLit: DecimalLit '.' DecimalLit Exponent?

| DecimalLit Exponent

| '.' DecimalLit Exponent?

;

fragment Escape: '\\' [tbnr"'\\];

fragment Char: ~[\\'"];

StringLit: '"' (Escape | Char)\* '"';

type: 'int'

| 'float'

| 'string'

| 'void'

| 'bool'

| 'graph'

| 'digraph'

;

edge: '[' source=StringLit ',' target=StringLit ']';

graphLit: gtype=('graph'|'digraph') '{' (edge (',' edge)\*)? '}';

arg: Identifier type;

argList: (arg (',' arg)\*)?;

atom: IntLit #Integer

| FloatLit #Float

| BoolLit #Boolean

| StringLit #String

| Identifier #VarName

| graphLit #Graph

| funcCall #FunctionCall

| type '(' expr ')' #Cast

;

expr: atom #Atomic

| <assoc=right> op=('-'|'+'|'!') expr #Unary

| '(' expr ')' #Assoc

| expr op=('\*'|'/'|'%') expr #MulDivMod

| expr op=('+'|'-') expr #AddSub

| expr op=('>'|'<'|'>='|'<=') expr #Comparison

| expr op=('=='|'!=') expr #Equality

| expr '&&' expr #LogicAnd

| expr '||' expr #LogicOr

;

exprList: (expr (',' expr)\*)? ;

glExpr: IntLit | FloatLit | BoolLit | StringLit ;

glVarDec: 'var' Identifier type ('=' glExpr)? ;

varDec: 'var' Identifier type ('=' expr)? ;

funcDef: 'func' Identifier '(' argList ')' type? '{' stmt\* '}' ;

funcCall: Identifier '(' exprList ')' ;

assignment: expr op=('='|'+='|'-='|'\*='|'/='|'%='|'&&='|'||=') expr ;

ifc: 'if' '(' expr ')' '{' stmt\* '}' elifc\* elsec? ;

elifc: 'elif' '(' expr ')' '{' stmt\* '}' ;

elsec: 'else' '{' stmt\* '}' ;

forc: 'for' '(' initial=assignment ';' cond=expr ';' mod=assignment ')' '{' loopStmt\* '}' ;

whilec: 'while' '(' expr ')' '{' loopStmt\* '}' ;

controlStmt: wr=('continue'|'break') ';' ;

returnStmt: 'return' expr? ;

compoundStmt: ifc

| forc

| whilec

;

simpleStmt: varDec

| assignment

| expr

| returnStmt

;

stmt: simpleStmt ';'

| compoundStmt

;

loopStmt: (stmt | controlStmt) ;

init: (funcDef | glVarDec ';')+;