```
CATEGORIES

expression;
definition;
statement;
type;

NODES

program -> string definition* functionCreation* functionDefinition* run;

functionCreation -> string;

varDefinition:definition -> string type;
structDefinition:definition -> string fieldDefinition*;
fieldDefinition:definition -> string type;
functionDefinition:definition -> string parameters:varDefinition* type?
locals:varDefinition* statement*;

print:statement -> expression*;
println:statement -> expression*;
read:statement -> expression*;
functionCallStatement:statement -> string expression*;
assignment:statement -> left:expression right:expression;
conditional:statement -> expression ifStatements:statement*
elseStatements:statement*;
loop:statement -> fromStatements:statement* expression loopStatements:statement*;
return:statement -> expression?;

run -> string expression*;

intType:type -> ;
```

```
realType:type -> ;

charType:type -> ;

arrayType:type -> int type;

structType:type -> string;

voidType:type -> ;


variable:expression -> string;

intLiteral:expression -> int;

realLiteral:expression -> float;

charLiteral:expression -> string;

functionCallExpression:expression -> string expression*;

structAccess:expression -> expr:expression string;

arrayAccess:expression -> left:expression right:expression;

cast:expression -> castType:type expression;


arithmeticBinary:expression -> left:expression operator:string right:expression;

arithmeticUnary:expression -> operator:string expr:expression;


logicBinary:expression -> left:expression operator:string right:expression;

logicUnary:expression -> operator:string expr:expression;


relationalBinary:expression -> left:expression operator:string right:expression;


ATTRIBUTE GRAMMAR Identification


variable -> varDefinition;

varDefinition -> scope:int;

functionCallStatement -> functionDefinition;

functionCallExpression -> functionDefinition;

run -> functionDefinition;

structType -> structDefinition;

fieldDefinition -> structDefinition;
```

```
ATTRIBUTE GRAMMAR TypeChecking

expression -> lvalue:boolean;
expression -> expressionType:type;
statement -> function:functionDefinition;
functionDefinition -> hasReturn:boolean;
structAccess -> fieldDefinition;

ATTRIBUTE GRAMMAR MemoryAllocation

varDefinition -> [inh] address:int;
structDefinition -> [inh] address:int;
fieldDefinition -> [inh] address:int;
functionDefinition -> [inh] address:int;

CODE SPECIFICATION Mapl

run[program]
execute[statement]
execute[run]
value[expression]
address[expression]

metadata[program]
metadata[varDefinition]
```