Universidad del Valle de Guatemala
Construcción de Compiladores
Sebastian Galindo 15452

Documentación de análisis Fase 1

Diagrama de Flujo

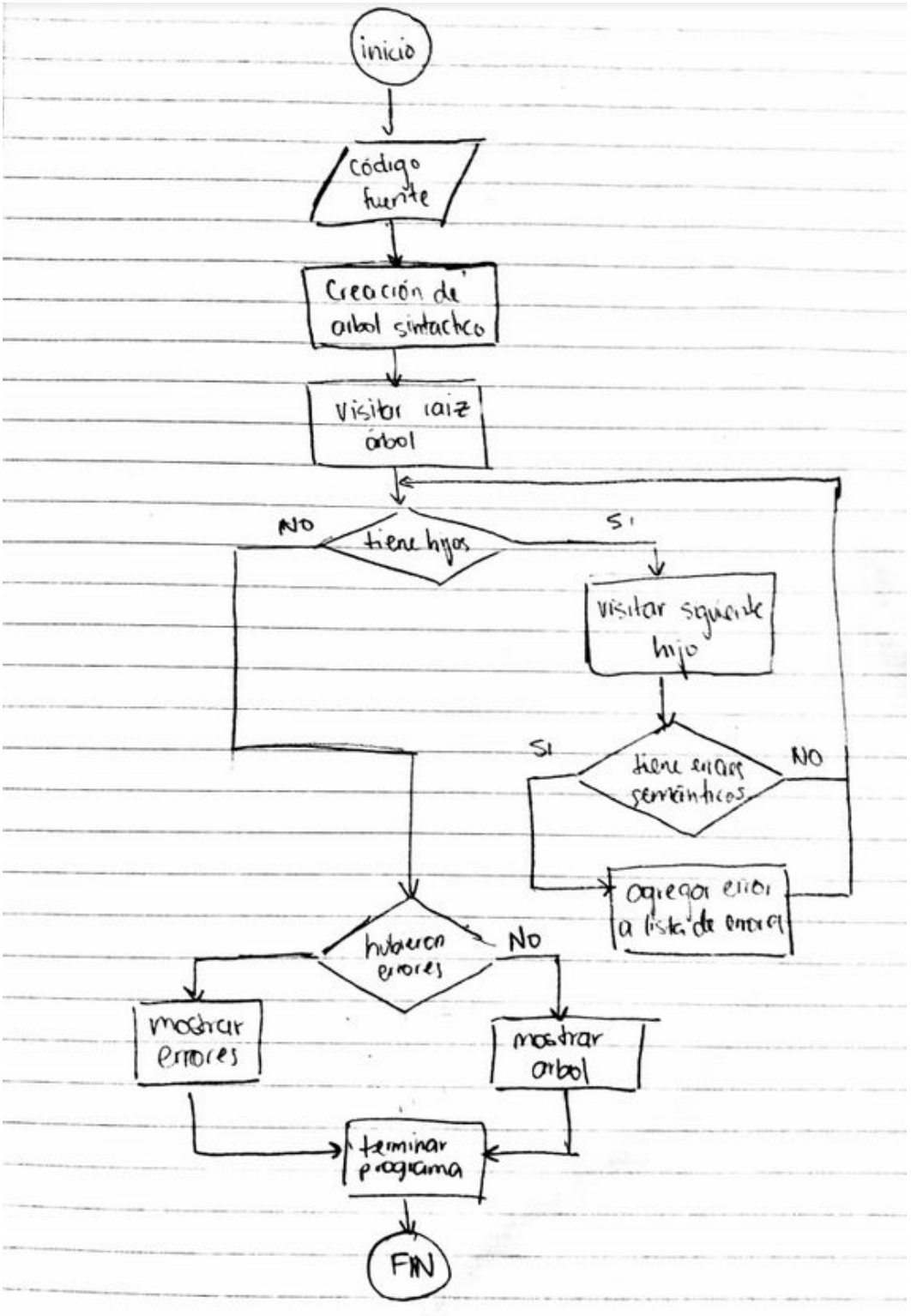
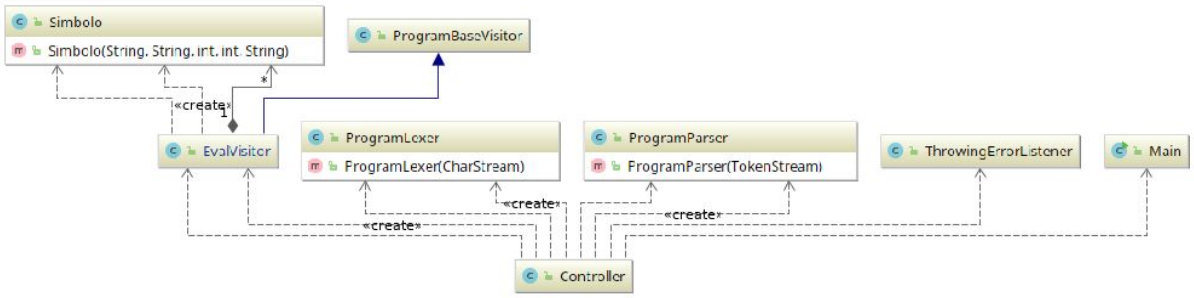


Diagrama de Clases



(El diagrama se mira simple pero es porque añadir los parámetros y atributos, por las clases propias de antlr, harían que fuera muy grande él UML)

Sistema de Tipos

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

program
  : 'class' ID '{' (declaration)* '}' ;

declaration
  : structDeclaration
  | varDeclaration
  | methodDeclaration ;

varDeclaration
  : varType ID ';'
  | varType ID '[' NUM ']' ';'     (varDeclaration.NUM() > 0)
  ;

structDeclaration
  : STRUCT ID '{' (varDeclaration)* finbloque
  ;

varType
  : INT | CHAR |  BOOLEAN |  STRUCT ID |  structDeclaration |  VOID ;

methodDeclaration
  : methodType ID '(' (parameter | (parameter (',' parameter)*)*) ')' block
  ;


methodType
  : INT              //(type = “int”)
  | CHAR           //(type = “char”)
  | BOOLEAN            //(type = “boolean”)
  | VOID          //(type = “void”)
  ;
```

```
parameter
  : parameterType ID
  | parameterType ID '['']'
  ;

parameterType
  : INT              //(type = “int”)
  | CHAR             //(type = “char”)
  | BOOLEAN          //(type = “boolean”)
  ;


block
  : '{' (varDeclaration)*(statement)* finbloque
  ;

finbloque
   :  '}'
   ;

statement
  : 'if' '(' expression ')' block (statementElse)? //(expression.type = “boolean”)
  | WHILE '(' expression ')' block              //(expression.type = “boolean”)
  | 'return' ( expression )? ';'               //(expression.type =  methodType.type)
  | methodCall ';'                          //(methodCall.type != “void”)
  | block
  | location '=' expression ';'                //(location.type = expression.type)
  |  locationArray '=' expression ';'         //(locationArray.type = expression.type)
  | (expression)';'
  ;

statementElse
   :  ELSE block ;

location
  :  (ID)('.' locationMember)? ;

locationMember
  :  (ID)('.' locationMember)?
   |   locationArray
  ;

locationArray
       :   ID '[' expression ']' ('.' locationMember)?  //(expression.type = “int”)

       ;
locationArray2
       :   ID '[' expression ']' ('.' locationMember)?

       ;
```

```
locationMethod
  : '.' locationMember
  ;

expression
 : andExpr
 | expression cond_op andExpr     //(expression.type = “boolean” && andExpr=”boolean”)
 ;

andExpr
 : eqExpr
 | andExpr cond_op eqExpr          //(andExpr.type = “boolean” && eqExpr.type=”boolean”)
 ;

eqExpr
 : relationExpr
 | eqExpr eq_op relationExpr      //(eqExpr.type == relationExpr.type)
 ;

relationExpr
 : addExpr
 | relationExpr rel_op addExpr    //(relationExpr.type = “int” && addExpr.type=”int”)
 ;

addExpr
 : multExpr
 | addExpr arith_op multExpr     //(addEpre.type = “int” && multExpr.type = “int”)
 ;

multExpr
 : unaryExpr
 | multExpr arith_op unaryExpr     (multExpr.type=”int” && unaryExpr.type = “int”)
 ;


unaryExpr
 :     '('(INT|CHAR)')'  value
 | '-' value
 | '!' value
 | value
 ;

value
 : location
  |  locationArray2
 | methodCall
 | literal
 | '(' expression ')'
 ;
```

```
methodCall
  : ID '(' (arg (',' arg)*)? ')'
  ;

arg
  : expression  ;

arith_op
  : '+'              #ao_plus
  | '-'              #ao_minus
  | '*'              #ao_mult
  | '/'              #ao_div
  | '%'               #ao_mod
  ;


rel_op
  : '<'              #relop_lower
  | '>'              #relop_greater
  | '<='              #relop_lowerEq
  | '>='              #relop_greaterEq
  ;

eq_op
  : '=='          #eqop_eq
  | '!='          #eqop_diff
  ;

cond_op
  : '||'            #condop_or
  | '&&'             #condop_and
  ;


literal
  : int_literal |  char_literal |  boolean_literal
  ;

int_literal
  : NUM ;          //(type = “char”)

char_literal
  : Char ;          //(type = “char”)

boolean_literal
  : 'true' | 'false' ;          //(type = “boolean”)
```

Reglas Semánticas Validadas

1. Ningún identificador es declarado dos veces en el mismo ámbito
2. Ningún identificador es utilizado antes de ser declarado.
3. El programa contiene una definición de un método main sin parámetros, en donde se empezará la ejecución del programa.
4. num en la declaración de un arreglo debe de ser mayor a 0.
5. El número y tipos de argumentos en la llamada a un método deben de ser los mismos que los argumentos formales, es decir las firmas deben de ser idénticas.
6. Si un método es utilizado en una expresión, este debe de devolver un resultado.
7. La instrucción return no debe de tener ningún valor de retorno, si el método se declara del tipo void.
8. Él valor de retorno de un método debe de ser del mismo tipo con que fue declarado él método.
9. Si se tiene la expresión id[<expr>], id debe de ser un arreglo y el tipo de <expr> debe de ser int.
10. El tipo de <expr> en la estructura if y while, debe de ser del tipo boolean.
11. Los tipos de operandos para los operadores <arith_op> y <rel_op> deben de ser int.
12. Los tipos de operandos para los operadores <eq_ops> deben de ser int, char o boolean, y además ambos operandos deben de ser del mismo tipo.
13. Los tipos de operandos para los operadores <cond_ops> y el operador ! deben de ser del tipo boolean
14. El valor del lado derecho de una asignación, debe de ser del mismo tipo que la locación del lado izquierdo.

Oportunidades de mejora

- Implementar un contador de líneas de manera que si se le muestra un error al usuario pueda identificar qué línea es de forma rápida.

Comentarios personales

- Me gustó ver cómo el proyecto va agarrando forma conforme uno va avanzando en cada fase.