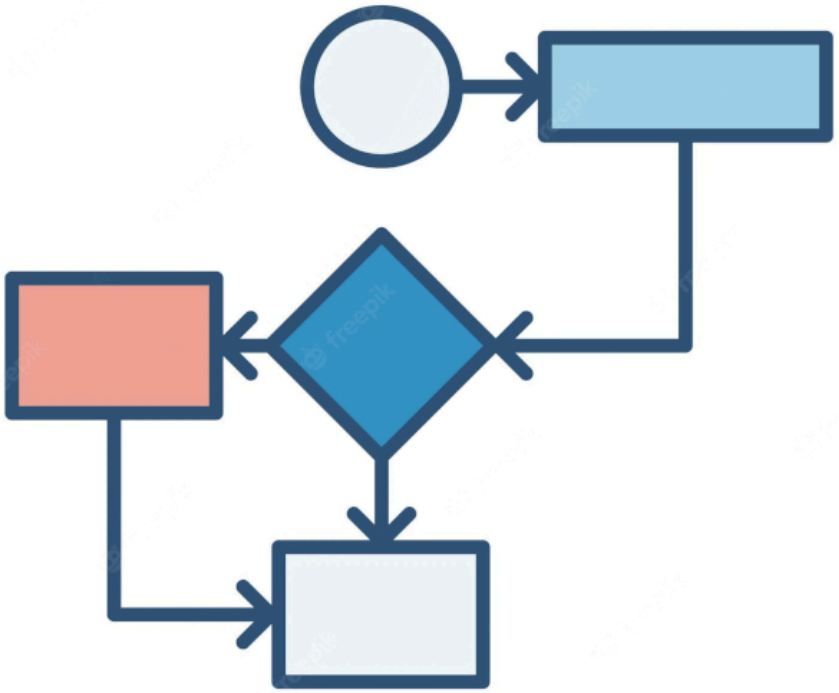
Manual Técnico

Traductor Pseudocódigo a Diagrama de Flujo



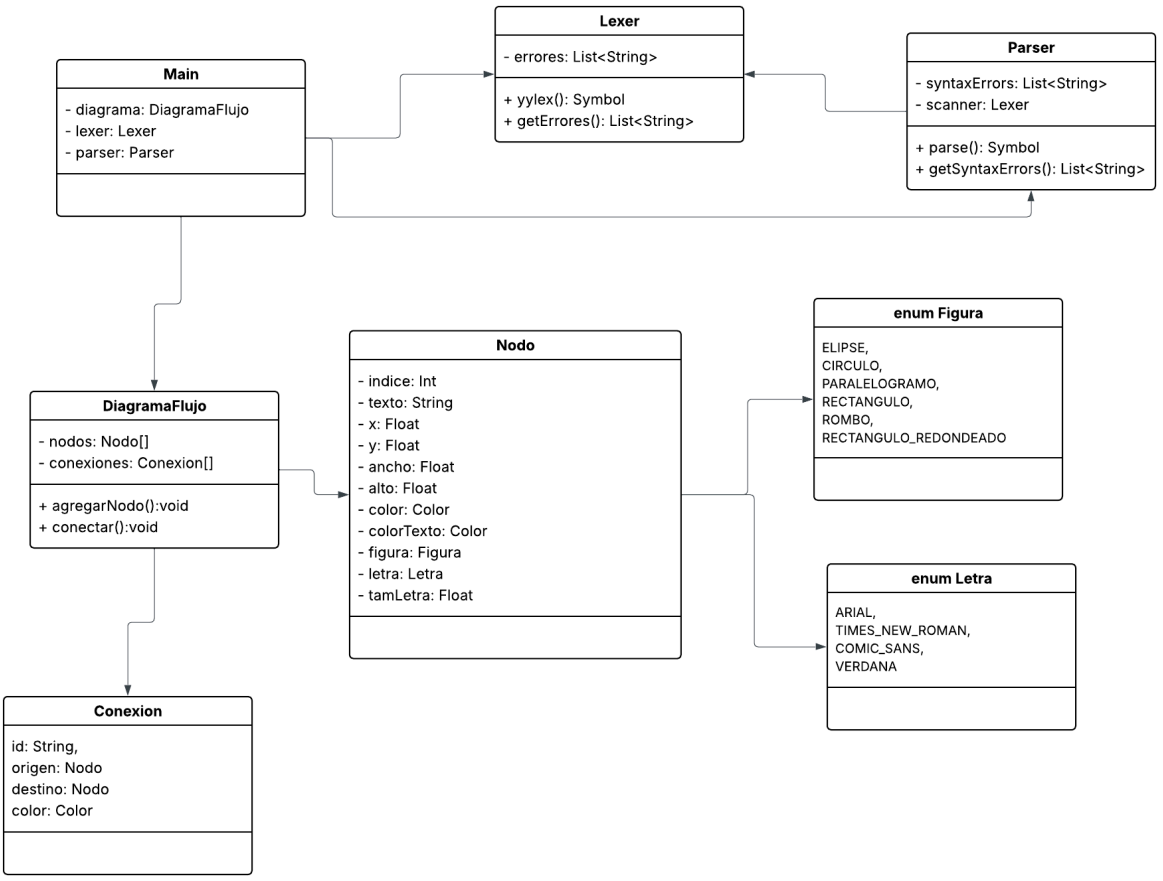
Introducción

Programa escrito en kotlin que traduce instrucciones en pseudocódigo a diagramas de flujo

Tecnologías utilizadas

open-jdk 21
jflex 1.9.1
java cup 11b
gradle 9.2.1
kotlin 2.2.20

Diagrama de clases



Análisis lexer

TOKENS

PALABRAS RESERVADAS:

INICIO
FIN
VAR
SI
ENTONCES
MIENTRAS
HACER
MOSTRAR
LEER

%DEFAULT
%COLOR_TEXTO_SI
%COLOR_SI
%FIGURA_SI
%LETRA_SI
%LETRA_SIZE_SI
%COLOR_TEXTO_MIENTRAS
%COLOR_MIENTRAS
%FIGURA_MIENTRAS
%LETRA_MIENTRAS
%LETRA_SIZE_MIENTRAS
%COLOR_TEXTO_BLOQUE
%COLOR_BLOQUE
%FIGURA_BLOQUE
%LETRA_BLOQUE
%LETRA_SIZE_BLOQUE

SEPARADOR:

SEPARADOR_SECCION (%%%%)

OPERADORES ARITMETICOS:

SUMA (+)
RESTA (-)
MULTIPLICACION (*)
DIVISION (/)

OPERADORES RELACIONALES:

OPERADOR_RELACIONAL (==, !=, >, <, >=, <=)

OPERADORES LOGICOS

OPERADOR_LOGICO(&&, ||, !)

SIMBOLOS ESPECIALES:

ASIGNACION (=)
TUBERIA (|)
COMA (,)
PARENTESIS_APERTURA(()
PARENTESIS_CIERRE())

LITERALES NUMERICAS:

ENTERO ([0-9]+)
DECIMAL ([0-9]+\.[0-9]+)

LITERALES DE TEXTO:

CADENA (\"([^\"\n]|\\.)*\")

IDENTIFICADOR:

IDENTIFICADOR ([a-zA-Z][a-zA-Z0-9_]*)

COMENTARIOS:

COMENTARIO_LINEA (\#.*)

NOMBRES DE FIGURAS:

NOMBRE_FIGURA (ELIPSE, CIRCULO, PARALELOGRAMO, RECTANGULO, ROMBO, RECTANGULO_REDONDEADO)

TIPOS DE LETRA

TIPO_LETRA (ARIAL, TIMES_NEW_ROMAN, COMIC_SANS, VERDANA)

COLORES:

COLOR_HEXADECIMAL (H[0-9A-Fa-f]{6})

ESPACIOS EN BLANCO:

--- ([\t\r\n]+)

ERROR:

ERROR ()

Análisis parser

Gramática formal:

G = (V, Σ, P, S)

V = {programa, seccion_algoritmo, seccion_configuracion, instrucciones, instruccion, bloque, condicion, expresion, config_instrucciones, config_instruccion, color, figura, tipo_letra}

Σ = {INICIO, FIN, VAR, SI, ENTONCES, MIENTRAS, HACER, MOSTRAR, LEER, ENTERO, DECIMAL, IDENTIFICADOR, CADENA, SUMA, RESTA, MULTIPLICACION, DIVISION, MAYOR, MENOR, MAYOR_IGUAL, MENOR_IGUAL, IGUAL, DIFERENTE, AND, OR, NOT, ASIGNACION, TUBERIA, COMA, PA, PC, SEPARADOR, DEFAULT_TOKEN, COLOR_TEXTO_SI, COLOR_SI, FIGURA_SI, LETRA_SI, LETRA_SIZE_SI, COLOR_TEXTO_MIENTRAS, COLOR_MIENTRAS, FIGURA_MIENTRAS, LETRA_MIENTRAS, LETRA_SIZE_MIENTRAS, COLOR_TEXTO_BLOQUE, COLOR_BLOQUE, FIGURA_BLOQUE, LETRA_BLOQUE, LETRA_SIZE_BLOQUE, ELIPSE, CIRCULO, PARALELOGRAMO, RECTANGULO, ROMBO, RECTANGULO_REDONDEADO, ARIAL, TIMES_NEW_ROMAN, COMIC_SANS, VERDANA, COLOR_HEX}

S = programa

P:
programa -> seccion_algoritmo SEPARADOR seccion_configuracion
seccion_algoritmo -> INICIO instrucciones FIN
instrucciones -> instrucciones instruccion | instruccion
instruccion -> VAR IDENTIFICADOR ASIGNACION expresion
| IDENTIFICADOR ASIGNACION expresion
| MOSTRAR expresion
| MOSTRAR CADENA
| LEER IDENTIFICADOR
| SI PA condicion PC ENTONCES bloque FIN SI
| MIENTRAS PA condicion PC HACER bloque FIN MIENTRAS
bloque -> instrucciones
condicion -> expresion MAYOR expresion
| expresion MENOR expresion
| expresion MAYOR_IGUAL expresion
| expresion MENOR_IGUAL expresion
| expresion IGUAL expresion
| expresion DIFERENTE expresion
| condicion AND condicion
| condicion OR condicion

| NOT condicion
| PA condicion PC
expresion -> expresion SUMA expresion
| expresion RESTA expresion
| expresion MULTIPLICACION expresion
| expresion DIVISION expresion
| PA expresion PC
| ENTERO
| DECIMAL
| IDENTIFICADOR
seccion_configuracion -> config_instrucciones
config_instrucciones -> config_instrucciones config_instruccion | config_instruccion
config_instruccion -> DEFAULT_TOKEN ASIGNACION expresion
| COLOR_TEXTO_SI ASIGNACION color TUBERIA expresion
| COLOR_SI ASIGNACION color TUBERIA expresion
| FIGURA_SI ASIGNACION figura TUBERIA expresion
| LETRA_SI ASIGNACION tipo_letra TUBERIA expresion
| LETRA_SIZE_SI ASIGNACION expresion TUBERIA expresion
| COLOR_TEXTO_MIENTRAS ASIGNACION color TUBERIA expresion
| COLOR_MIENTRAS ASIGNACION color TUBERIA expresion
| FIGURA_MIENTRAS ASIGNACION figura TUBERIA expresion
| LETRA_MIENTRAS ASIGNACION tipo_letra TUBERIA expresion
| LETRA_SIZE_MIENTRAS ASIGNACION expresion TUBERIA expresion
| COLOR_TEXTO_BLOQUE ASIGNACION color TUBERIA expresion
| COLOR_BLOQUE ASIGNACION color TUBERIA expresion
| FIGURA_BLOQUE ASIGNACION figura TUBERIA expresion
| LETRA_BLOQUE ASIGNACION tipo_letra TUBERIA expresion
| LETRA_SIZE_BLOQUE ASIGNACION expresion TUBERIA expresion
color -> COLOR_HEX | expresion COMA expresion COMA expresion
figura -> ELIPSE | CIRCULO | PARALELOGRAMO | RECTANGULO | ROMBO |
RECTANGULO_REDONDEADO
tipo_letra -> ARIAL | TIMES_NEW_ROMAN | COMIC_SANS | VERDANA

Archivo configuración flex

```
%%

%public
%class Lexer
%unicode
%line
%column
%cup

%{
```

```
import java_cup.runtime.Symbol;
import java.io.*;
import java.util.LinkedList;

private LinkedList<String> listaErrores = new LinkedList<>();

private void agregarError(String lexema) {
        listaErrores.add(
        "Error léxico -> " + lexema +
        " Línea: " + (yyline + 1) +
        " Columna: " + (yycolumn + 1)
        );
}

public LinkedList<String> getErrores(){
        return listaErrores;
}

private Symbol symbol(int type){
        return new Symbol(type, yyline+1, yycolumn+1, yytext());
}

private Symbol symbol(int type, Object value){
        return new Symbol(type, yyline+1, yycolumn+1, value);
}

%}

ENTERO              = [0-9]+
DECIMAL             = [0-9]+\.[0-9]+
IDENTIFICADOR       = [a-zA-Z][a-zA-Z0-9_]*
CADENA              = \"([^\"\n]|\\.)*\"
COMENTARIO_LINEA          = \#.*
ESPACIOS            = [ \t\r\n]+
COLOR_HEX           = H[0-9A-Fa-f]{6}
SEPARADOR           = "%%%%"

%%

{SEPARADOR}         { return symbol(sym.SEPARADOR); }

"INICIO"        { return symbol(sym.INICIO); }
"FIN"           { return symbol(sym.FIN); }
"VAR"           { return symbol(sym.VAR); }
"SI"            { return symbol(sym.SI); }
"ENTONCES"          { return symbol(sym.ENTONCES); }
"MIENTRAS"          { return symbol(sym.MIENTRAS); }
"HACER"       { return symbol(sym.HACER); }
```

```
"MOSTRAR"            { return symbol(sym.MOSTRAR); }
"LEER"          { return symbol(sym.LEER); }

"%DEFAULT"                  { return symbol(sym.DEFAULT); }
"%COLOR_TEXTO_SI"                   { return symbol(sym.COLOR_TEXTO_SI); }
"%COLOR_SI"                 { return symbol(sym.COLOR_SI); }
"%FIGURA_SI"                { return symbol(sym.FIGURA_SI); }
"%LETRA_SI"                 { return symbol(sym.LETRA_SI); }
"%LETRA_SIZE_SI"            { return symbol(sym.LETRA_SIZE_SI); }

"%COLOR_TEXTO_MIENTRAS"     { return symbol(sym.COLOR_TEXTO_MIENTRAS); }
"%COLOR_MIENTRAS"           { return symbol(sym.COLOR_MIENTRAS); }
"%FIGURA_MIENTRAS"          { return symbol(sym.FIGURA_MIENTRAS); }
"%LETRA_MIENTRAS"           { return symbol(sym.LETRA_MIENTRAS); }
"%LETRA_SIZE_MIENTRAS"      { return symbol(sym.LETRA_SIZE_MIENTRAS); }

"%COLOR_TEXTO_BLOQUE"       { return symbol(sym.COLOR_TEXTO_BLOQUE); }
"%COLOR_BLOQUE"             { return symbol(sym.COLOR_BLOQUE); }
"%FIGURA_BLOQUE"            { return symbol(sym.FIGURA_BLOQUE); }
"%LETRA_BLOQUE"          { return symbol(sym.LETRA_BLOQUE); }
"%LETRA_SIZE_BLOQUE"        { return symbol(sym.LETRA_SIZE_BLOQUE); }

"ELIPSE"            { return symbol(sym.ELIPSE); }
"CIRCULO"           { return symbol(sym.CIRCULO); }
"PARALELOGRAMO"          { return symbol(sym.PARALELOGRAMO); }
"RECTANGULO"                { return symbol(sym.RECTANGULO); }
"ROMBO"             { return symbol(sym.ROMBO); }
"RECTANGULO_REDONDEADO"  { return symbol(sym.RECTANGULO_REDONDEADO); }

"ARIAL"         { return symbol(sym.ARIAL); }
"TIMES_NEW_ROMAN"       { return symbol(sym.TIMES_NEW_ROMAN); }
"COMIC_SANS"        { return symbol(sym.COMIC_SANS); }
"VERDANA"           { return symbol(sym.VERDANA); }

"+" { return symbol(sym.SUMA); }
"-" { return symbol(sym.RESTA); }
"*" { return symbol(sym.MULTIPLICACION); }
"/" { return symbol(sym.DIVISION); }

">=" { return symbol(sym.MAYOR_IGUAL); }
"<=" { return symbol(sym.MENOR_IGUAL); }
"==" { return symbol(sym.IGUAL); }
"!=" { return symbol(sym.DIFERENTE); }
">"  { return symbol(sym.MAYOR); }
"<"  { return symbol(sym.MENOR); }

"&&" { return symbol(sym.AND); }
"||" { return symbol(sym.OR); }
```

```
"!"  { return symbol(sym.NOT); }

"=" { return symbol(sym.ASIGNACION); }
"|" { return symbol(sym.TUBERIA); }
"," { return symbol(sym.COMA); }
"(" { return symbol(sym.PA); }
")" { return symbol(sym.PC); }

{DECIMAL}   { return symbol(sym.DECIMAL, yytext()); }
{ENTERO}      { return symbol(sym.ENTERO, yytext()); }
{CADENA}      { return symbol(sym.CADENA, yytext()); }
{COLOR_HEX} { return symbol(sym.COLOR_HEX, yytext()); }
{IDENTIFICADOR} { return symbol(sym.IDENTIFICADOR, yytext()); }

{COMENTARIO_LINEA}  { }
{ESPACIOS}              { }

<<EOF>> { return new Symbol(sym.EOF); }

. {
        agregarError(yytext());
}
```

Archivo de configuración cup

```
import java_cup.runtime.*;
import java.util.*;
import tu.paquete.*;

parser code {:

        Lexer lex;
        private List<String> syntaxErrors;

        public Parser(Lexer lex){
        super(lex);
        this.lex = lex;
        syntaxErrors = new ArrayList<>();
        }

        public List<String> getSyntaxErrors(){
        return this.syntaxErrors;
        }

        public void syntax_error(Symbol cur_token){
        syntaxErrors.add(
        "Error sintactico: " + cur_token.value +
```

```
        " linea " + cur_token.left +
        " columna " + cur_token.right
        );
        }

        public void unrecovered_syntax_error(Symbol cur_token){
        syntaxErrors.add("Error que no se puede recuperar");
        }

:};


terminal INICIO, FIN, VAR, SI, ENTONCES, MIENTRAS, HACER;
terminal MOSTRAR, LEER;

terminal ENTERO, DECIMAL, IDENTIFICADOR, CADENA;

terminal SUMA, RESTA, MULTIPLICACION, DIVISION;
terminal MAYOR, MENOR, MAYOR_IGUAL, MENOR_IGUAL, IGUAL, DIFERENTE;

terminal AND, OR, NOT;

terminal ASIGNACION;
terminal TUBERIA, COMA;
terminal PA, PC;

terminal SEPARADOR;

terminal DEFAULT_TOKEN,
        COLOR_TEXTO_SI, COLOR_SI, FIGURA_SI, LETRA_SI, LETRA_SIZE_SI,
        COLOR_TEXTO_MIENTRAS, COLOR_MIENTRAS, FIGURA_MIENTRAS,
LETRA_MIENTRAS, LETRA_SIZE_MIENTRAS,
        COLOR_TEXTO_BLOQUE, COLOR_BLOQUE, FIGURA_BLOQUE,
LETRA_BLOQUE, LETRA_SIZE_BLOQUE;

terminal ELIPSE, CIRCULO, PARALELOGRAMO, RECTANGULO, ROMBO,
RECTANGULO_REDONDEADO;
terminal ARIAL, TIMES_NEW_ROMAN, COMIC_SANS, VERDANA;

terminal COLOR_HEX;


non terminal Programa programa;
non terminal List seccion_algoritmo;
non terminal List seccion_configuracion;

non terminal List instrucciones;
non terminal Instruccion instruccion;
```

```
non terminal List bloque;

non terminal String condicion;
non terminal String expresion;

non terminal List config_instrucciones;
non terminal Configuracion config_instruccion;

non terminal Object color, figura, tipo_letra;

precedence left OR;
precedence left AND;
precedence left IGUAL, DIFERENTE;
precedence left MAYOR, MENOR, MAYOR_IGUAL, MENOR_IGUAL;
precedence left SUMA, RESTA;
precedence left MULTIPLICACION, DIVISION;
precedence right NOT;


start with programa;


programa ::= seccion_algoritmo:alg SEPARADOR seccion_configuracion:conf
        {: RESULT = new Programa(alg, conf); :}
;

seccion_algoritmo ::= INICIO instrucciones:lista FIN
        {: RESULT = lista; :}
;

instrucciones ::=
        instrucciones:lista instruccion:inst
        {:
        lista.add(inst);
        RESULT = lista;
        :}
        |   instruccion:inst
        {:
        List nueva = new LinkedList();
        nueva.add(inst);
        RESULT = nueva;
        :}
;

instruccion ::=

        VAR IDENTIFICADOR:id ASIGNACION expresion:exp
        {:
```

```
RESULT = new Instruccion(
        TipoInstruccion.ASIGNACION,
        "VAR " + id,
        exp
);
:}

| IDENTIFICADOR:id ASIGNACION expresion:exp
{:
RESULT = new Instruccion(
        TipoInstruccion.ASIGNACION,
        id.toString(),
        exp
);
:}

| MOSTRAR expresion:exp
{:
RESULT = new Instruccion(
        TipoInstruccion.MOSTRAR,
        exp,
        null
);
:}

| MOSTRAR CADENA:cad
{:
RESULT = new Instruccion(
        TipoInstruccion.MOSTRAR,
        cad.toString(),
        null
);
:}

| LEER IDENTIFICADOR:id
{:
RESULT = new Instruccion(
        TipoInstruccion.LEER,
        id.toString(),
        null
);
:}

| SI PA condicion:cond PC ENTONCES bloque:bl FIN SI
{:
Instruccion inst =
        new Instruccion(TipoInstruccion.SI, cond, null);
inst.setBloque(bl);
```

```
        RESULT = inst;
        :}

        | MIENTRAS PA condicion:cond PC HACER bloque:bl FIN MIENTRAS
        {:
        Instruccion inst =
                new Instruccion(TipoInstruccion.MIENTRAS, cond, null);
        inst.setBloque(bl);
        RESULT = inst;
        :}
;

bloque ::= instrucciones:lista
        {: RESULT = lista; :}
;

condicion ::=
        expresion:e1 MAYOR expresion:e2             {: RESULT = e1 + " > " + e2; :}
        |  expresion:e1 MENOR expresion:e2          {: RESULT = e1 + " < " + e2; :}
        |  expresion:e1 MAYOR_IGUAL expresion:e2  {: RESULT = e1 + " >= " + e2; :}
        |  expresion:e1 MENOR_IGUAL expresion:e2  {: RESULT = e1 + " <= " + e2; :}
        |  expresion:e1 IGUAL expresion:e2          {: RESULT = e1 + " == " + e2; :}
        |  expresion:e1 DIFERENTE expresion:e2   {: RESULT = e1 + " != " + e2; :}
        |  condicion:c1 AND condicion:c2            {: RESULT = c1 + " && " + c2; :}
        |  condicion:c1 OR condicion:c2      {: RESULT = c1 + " || " + c2; :}
        |  NOT condicion:c                   {: RESULT = "!" + c; :}
        |  PA condicion:c PC                 {: RESULT = "(" + c + ")"; :}
;

expresion ::=
        expresion:e1 SUMA expresion:e2              {: RESULT = e1 + " + " + e2; :}
        |  expresion:e1 RESTA expresion:e2          {: RESULT = e1 + " - " + e2; :}
        |  expresion:e1 MULTIPLICACION expresion:e2 {: RESULT = e1 + " * " + e2; :}
        |  expresion:e1 DIVISION expresion:e2       {: RESULT = e1 + " / " + e2; :}
        |  PA expresion:e PC                 {: RESULT = "(" + e + ")"; :}
        |  ENTERO:num                        {: RESULT = num.toString(); :}
        |  DECIMAL:dec                       {: RESULT = dec.toString(); :}
        |  IDENTIFICADOR:id                  {: RESULT = id.toString(); :}
;

seccion_configuracion ::= config_instrucciones:lista
        {: RESULT = lista; :}
;

config_instrucciones ::=
        config_instrucciones:lista config_instruccion:conf
        {:
        lista.add(conf);
```

```
        RESULT = lista;
        :}
        |   config_instruccion:conf
        {:
        List nueva = new LinkedList();
        nueva.add(conf);
        RESULT = nueva;
        :}
;

config_instruccion ::=

        DEFAULT_TOKEN ASIGNACION expresion:ind
        {: RESULT = new Configuracion("DEFAULT", null, ind); :}

        | COLOR_TEXTO_SI ASIGNACION color:c TUBERIA expresion:ind
        {: RESULT = new Configuracion("COLOR_TEXTO_SI", c, ind); :}

        | COLOR_SI ASIGNACION color:c TUBERIA expresion:ind
        {: RESULT = new Configuracion("COLOR_SI", c, ind); :}

        | FIGURA_SI ASIGNACION figura:f TUBERIA expresion:ind
        {: RESULT = new Configuracion("FIGURA_SI", f, ind); :}

        | LETRA_SI ASIGNACION tipo_letra:l TUBERIA expresion:ind
        {: RESULT = new Configuracion("LETRA_SI", l, ind); :}

        | LETRA_SIZE_SI ASIGNACION expresion:val TUBERIA expresion:ind
        {: RESULT = new Configuracion("LETRA_SIZE_SI", val, ind); :}

        | COLOR_TEXTO_MIENTRAS ASIGNACION color:c TUBERIA expresion:ind
        {: RESULT = new Configuracion("COLOR_TEXTO_MIENTRAS", c, ind); :}

        | COLOR_MIENTRAS ASIGNACION color:c TUBERIA expresion:ind
        {: RESULT = new Configuracion("COLOR_MIENTRAS", c, ind); :}

        | FIGURA_MIENTRAS ASIGNACION figura:f TUBERIA expresion:ind
        {: RESULT = new Configuracion("FIGURA_MIENTRAS", f, ind); :}

        | LETRA_MIENTRAS ASIGNACION tipo_letra:l TUBERIA expresion:ind
        {: RESULT = new Configuracion("LETRA_MIENTRAS", l, ind); :}

        | LETRA_SIZE_MIENTRAS ASIGNACION expresion:val TUBERIA expresion:ind
        {: RESULT = new Configuracion("LETRA_SIZE_MIENTRAS", val, ind); :}

        | COLOR_TEXTO_BLOQUE ASIGNACION color:c TUBERIA expresion:ind
        {: RESULT = new Configuracion("COLOR_TEXTO_BLOQUE", c, ind); :}
```

```
        | COLOR_BLOQUE ASIGNACION color:c TUBERIA expresion:ind
        {: RESULT = new Configuracion("COLOR_BLOQUE", c, ind); :}

        | FIGURA_BLOQUE ASIGNACION figura:f TUBERIA expresion:ind
        {: RESULT = new Configuracion("FIGURA_BLOQUE", f, ind); :}

        | LETRA_BLOQUE ASIGNACION tipo_letra:l TUBERIA expresion:ind
        {: RESULT = new Configuracion("LETRA_BLOQUE", l, ind); :}

        | LETRA_SIZE_BLOQUE ASIGNACION expresion:val TUBERIA expresion:ind
        {: RESULT = new Configuracion("LETRA_SIZE_BLOQUE", val, ind); :}
;


color ::=
        COLOR_HEX:hex
        {: RESULT = hex.toString(); :}
        |   expresion:r COMA expresion:g COMA expresion:b
        {: RESULT = r + "," + g + "," + b; :}
;

figura ::=
        ELIPSE:e
        {: RESULT = e.toString(); :}
        |   CIRCULO:c
        {: RESULT = c.toString(); :}
        |   PARALELOGRAMO:p
        {: RESULT = p.toString(); :}
        |   RECTANGULO:r
        {: RESULT = r.toString(); :}
        |   ROMBO:ro
        {: RESULT = ro.toString(); :}
        |   RECTANGULO_REDONDEADO:rr
        {: RESULT = rr.toString(); :}
;

tipo_letra ::=
        ARIAL:a
        {: RESULT = a.toString(); :}
        |   TIMES_NEW_ROMAN:t
        {: RESULT = t.toString(); :}
        |   COMIC_SANS:c
        {: RESULT = c.toString(); :}
        |   VERDANA:v
        {: RESULT = v.toString(); :}
;
```