



GRAMATICAS

Proyecto No. 2
LABORATORIO ORGANIZACION DE LENGUAJES Y
COMPILADORES 1 Sección C

José Eduardo Galdámez González
Carne: 202109732

Gramáticas:

AREA DE IMPORTACIONES:

```
%{
let Raiz = require("../Entorno/Raiz").Raiz;
let Tipo = require("../Entorno/Simbolos/Tipo").Tipo;
let TipoPrimitivo = require("../Entorno/Simbolos/TipoPrimitivo").TipoPrimitivo;
let DeclararVariable = require("../Instrucciones/DeclararVariable").DeclararVariable;
let DeclararFuncion = require("../Instrucciones/DeclararFuncion").DeclararFuncion;
let DeclararArreglo = require("../Instrucciones/DeclararArreglo").DeclararArreglo;
let DeclararLista = require("../Instrucciones/DeclararLista").DeclararLista;
let Asignacion = require("../Instrucciones/Asignacion").Asignacion;
let AsignacionVector = require("../Instrucciones/AsignacionVector").AsignacionVector;
let Ternario = require("../Expresiones/Ternario").Ternario;
let If = require("../Instrucciones/If").If;
let FuncionLenguaje = require("../Expresiones/FuncionLenguaje").FuncionLenguaje;
let AccesoVariable = require("../Expresiones/AccesoVariable").AccesoVariable;
let AccesoLista = require("../Expresiones/AccesoLista").AccesoLista;
let AccesoVector = require("../Expresiones/AccesoVector").AccesoVector;
let LlamadaFuncion = require("../Expresiones/LlamadaFuncion").LlamadaFuncion;
let LlamadaPrint = require("../Expresiones/LlamadaPrint").LlamadaPrint;
let OperacionAritmetica = require("../Expresiones/OperacionAritmetica").OperacionAritmetica;
let OperacionLogica = require("../Expresiones/OperacionLogica").OperacionLogica;
let OperacionRelacional = require("../Expresiones/OperacionRelacional").OperacionRelacional;
let While = require("../Instrucciones/While").While;
let ReturnPR = require("../Expresiones/ReturnPR").ReturnPR;
let Break = require("../Expresiones/Break").Break;
let Continue = require("../Expresiones/Continue").Continue;
let Valor = require("../Expresiones/Valor").Valor;
let Incremento = require("../Instrucciones/Incremento").Incremento;
let Decremento = require("../Instrucciones/Decremento").Decremento;
let Switch = require("../Instrucciones/Switch").Switch;
let CaseSwitch = require("../Instrucciones/CaseSwitch").CaseSwitch;
let For = require("../Instrucciones/For").For;
let DoWhile = require("../Instrucciones/DoWhile").DoWhile;
let Casteo = require("../Expresiones/Casteo").Casteo;
let InsertarLista = require("../Instrucciones/InsertarLista").InsertarLista;
let ModificarLista = require("../Instrucciones/ModificarLista").ModificarLista;
let Error = require("../Tabla/Error").Error;
let TablaError = require("../Tabla/TablaError").TablaError;
}%
```

AREA DE EXPRESIONES REGULARES:

```
%lex
%options case-sensitive

digit [0-9]
cor1  "["
cor2  "]"
esc   "\\\"
int   (?:[0-9]|[1-9][0-9]+)
exp   (?:[eE][+-]?[0-9]+)
frac  (?:\.[0-9]+)

%%

\s+ /* skip whitespace */
<<EOF>> {return 'EOF';}

/* COMENTARIOS */
"//".* /* IGNORE */
[/][*][^*]*[+](?:[/][^*]*[+])*[/] /* IGNORE */
```

Luego se definieron las expresiones regulares de los datos primitivos del lenguaje:

```
/* ===== EXPRESIONES REGULARES ===== */
([a-zA-ZÑñ]|("_"[a-zA-ZÑñ]))([a-zA-ZÑñ]|[0-9]|"_"*) yytext = yytext.toLowerCase(); return 'id';
\"(?:{cor1}|{cor2}|["\\"]|["b"r"t"/["\\"]]|["^["\\"]])*\" yytext = yytext.substr(1,yytext.length-2); return 'cadena';
\\'?(?:{esc}["b"r"t"/{esc}]|{esc}"u"[a-zA-Z0-9]{4}|["^{"esc}"]\\' yytext = yytext.substr(1,yytext.length-2); return 'caracter';
{int}{frac}\\b return 'decimal';
{int}\\b return 'entero';

/Error return 'entero'
```

PALABRAS RESERVADAS

```
/* ===== PALABRAS RESERVADAS ===== */
"true"           { return 'ttrue';    }
"false"          { return 'tfalse';   }
"int"            { return 'tinteger'; }
"boolean"        { return 'tboolean'; }
"double"         { return 'tdouble';  }
"String"         { return 'tstring';  }
"char"           { return 'tchar';    }
"if"             { return 'tif';       }
"while"          { return 'twhile';   }
"for"            { return 'tfor';      }
"else"           { return 'telse';     }
"void"           { return 'tvoid';     }
"return"         { return 'treturn';   }
"new"            { return 'tnew';      }
"do"             { return 'tdo';       }
"list"           { return 'tlist';     }
"add"            { return 'tadd';      }
"switch"         { return 'tswitch';   }
"case"           { return 'tcase';     }
"default"        { return 'tdefault';  }
"toLowerCase"    { return 'ttoLower';  }
"toUpperCase"    { return 'ttoUpper';  }
"truncate"       { return 'ttruncate'; }
"round"          { return 'tround';    }
"length"         { return 'tlength';   }
"typeof"         { return 'ttypeof';   }
"toString"       { return 'ttoString'; }
"toCharArray"    { return 'ttoCharArray'; }
"main"           { return 'tmain';     }
"print"          { return 'tPrint';    }
"break"          { return 'tBreak';    }
"continue"       { return 'tContinue'; }
```

ÁREA DE SÍMBOLOS

```
"."           {return '.';}
"++"          {return '++';}
"--"          {return '--';}
"+"           {return '+';}
"_"           {return '-';}
"*"           {return '*';}
"/"           {return '/';}
"^"           {return '^';}
"%"           {return '%';}
"("           {return '(';}
")"           {return ')';}
"=="          {return '==';}
"!="          {return '!=';}
","           {return ',';}
";"           {return ';';}
":"           {return ':';}
"."           {return '.';}
"? "          {return '?';}
"||"          {return '||';}
"&&"          {return '&&';}
"!="          {return '!=';}
"! "          {return '!';}
"<="          {return '<=';}
">="          {return '>=';}
">"           {return '>';}
"<"           {return '<';}
"{"           {return '{';}
"}"           {return '}';}
"["           {return '[';}
"]"           {return ']';}
. { TablaError.insertarError(new Error("Lexico", `El caracter: "${yytext}" no pertenece dentro del lenguaje`, yylloc.first_line,yylloc.first_column));
  console.log("Lexico"+yytext+" "+ yylloc.first_line+" "+yylloc.first_column) }
```

/lex

```

/* ===== ASOCIATIVIDAD y PRECEDENCIA DE OPERADORES =====
/*Operaciones logicas*/
%left '||'
%left '&&'
%left '?'
%left ':'
%left '++' '--'
%left '!=' '==' '==='
%left '>' '<' '<=' '>='

/*Operaciones numericas*/
%left '+' '-'
%left '*' '/' '%'
%right '^'
%right negativo '!' '('

```

Acá se determina la precedencia de cada símbolo. Es decir, para realizar operaciones como lo son las aritméticas existen ciertas reglas para obtener el resultado correcto, entonces, podemos suponer que las precedencias determinan las reglas que se deben seguir para el buen comprendimiento de las entradas.

ANÁLISIS SINTÁCTICO

La gramática cuenta con una producción llamada INICIO la cual indica que un nuevo análisis ha comenzado. Según la gramática pueden venir sentencias o bien, podemos toparnos con un archivo vacío.

Una sentencia es básicamente una instrucción dentro del lenguaje. Estas son el corazón del lenguaje, ya que sin ellas no se ejecutaría ninguna instrucción y por lo tanto, no habría funcionalidad.

A continuación, se mostrará cada una de las producciones creadas y un ejemplo del tipo de entrada que permiten:

```

%start INICIO

%% /* language grammar */

INICIO
: SENTENCIAS EOF
{
    console.log("Parse de Jison entrada: OK ");
    let raiz = new Raiz($1);
    $$ = raiz;
    return raiz;
}
;

SENTENCIAS : SENTENCIAS SENTENCIA
{
    $1.push($2);
    $$ = $1;
}
| SENTENCIA
{
    let lstsent = [];
    lstsent.push($1);
    $$ = lstsent;
}
| error { TablaError.insertarError(new Error("Sintactico", `El error parte tras: "${yytext}" no acorde a la gramatica.`),this._$.first_line, this._$.first_column));}
;

BLOQUE_SENTENCAS : '{' SENTENCIAS '}'
{
    $$ = $2;
}
| '{' '}'
{
    $$ = [];
}
| error '}' { TablaError.insertarError(new Error("Sintactico", "Falta en el bloque de sentencias {}"),this._$.first_line, this._$.first_column));}
;

```



```

SENTENCIA :   DECLARACION ';'          { $$ = $1; }
              |   FUNCION                { $$ = $1; }
              |   LISTA_AGREGAR           { $$ = $1; }
              |   LISTA_MODIFICAR ';'     { $$ = $1; }
              |   ASIGNACION              { $$ = $1; }
              |   VECTOR_ASIGNAR          { $$ = $1; }
              |   IF                      { $$ = $1; }
              |   LLAMADA_FUNCION ';'     { $$ = $1; }
              |   WHILE                   { $$ = $1; }
              |   FOR                     { $$ = $1; }
              |   DO_WHILE                { $$ = $1; }
              |   INCREMENTO ';'          { $$ = $1; }
              |   DECREMENTO ';'          { $$ = $1; }
              |   PRINT ';'               { $$ = $1; }
              |   MAIN ';'                { $$ = $1; }
              |   RETURN                  { $$ = $1; }
              |   BREAK                    { $$ = $1; }
              |   CONTINUE                 { $$ = $1; }
              |   SWITCH                   { $$ = $1; }
;

```

MAIN: Inicia la ejecución del programa. Recibe la palabra reservada “main” y ejecuta una LLAMADA_FUNCION.

```

MAIN : tmain LLAMADA_FUNCION    { $$ = $2; }
;

```

PRINT: Imprime en pantalla cualquier tipo de expresión. Recibe la palabra reservada “print” seguido de un paréntesis que abre, una lista de expresiones y un paréntesis que cierra.

```

PRINT : tPrint '(' LISTA_EXP ')' { $$ = new LlamadaPrint($1, $3, @1.first_line, @1.first_column); }
;

```

DECLARACIÓN: Inicializa una variable de cualquier tipo. Además de listas y vectores con los distintos tipos de declaración que poseen.

```

DECLARACION : TIPO id '=' EXP
{
    $$ = new DeclararVariable($1, $2, $4, @2.first_line, @2.first_column);
}
| TIPO id
{
    $$ = new DeclararVariable($1, $2, undefined, @2.first_line, @2.first_column);
}
| TIPO '[' ']' id '=' tnew TIPO '[' EXP ']'
{
    $$ = new DeclararArreglo($1, $4, $7, undefined, $9, @2.first_line, @2.first_column);
}
| TIPO '[' ']' id '=' '(' LISTA_EXP ')'
{
    $$ = new DeclararArreglo($1, $4, undefined, $7, undefined, @2.first_line, @2.first_column);
}
| tlist '<' TIPO '>' id '=' tnew tlist '<' TIPO '>'
{
    $$ = new DeclararLista($3, $5, $10, undefined, @2.first_line, @2.first_column);
}
| tlist '<' TIPO '>' id '=' EXP
{
    $$ = new DeclararLista($3, $5, undefined, $7, @2.first_line, @2.first_column);
}
;

```

ASIGNACIÓN: Permite asignar o reasignar el valor que contiene una variable. Recibe el id de la variable, el “=” y una expresión que indica lo que podría ser.

LISTA/VECTOR ASIGNAR: Permiten agregar un valor a una lista o vector.

```
ASIGNACION      :   id '=' EXP ';'
                  {
                    $$ = new Asignacion($1, $3, @1.first_line, @1.first_column);
                  }
;

VECTOR_ASIGNAR  :   id '[' EXP ']' '=' EXP ';'
                  {
                    $$ = new AsignacionVector($1, $6, $3, @1.first_line, @1.first_column);
                  }
;

LISTA_AGREGAR   :   id '.' tadd '(' EXP ')' ';'
                  {
                    $$ = new InsertarLista($1, $5, @1.first_line, @1.first_column);
                  }
;
```

Condicional (IF/ELSE/ELSE IF): Permiten agregar un valor a una lista o vector

```
IF              :   tif '(' EXP ')' BLOQUE_SENTENCAS
                  {
                    $$ = new If($3, $5, [], @1.first_line, @1.first_column);
                  }
|
|   tif '(' EXP ')' BLOQUE_SENTENCAS ELSE
|   {
|       $$ = new If($3, $5, $6, @1.first_line, @1.first_column);
|   }
;

ELSE            :   telse IF
                  {
                    let else_sent = [];
                    else_sent.push($2);
                    $$ = else_sent;
                  }
|
|   telse BLOQUE_SENTENCAS
|   {
|       $$ = $2;
|   }
;
```

Ciclos:

```
DO_WHILE : tdo BLOQUE_SENTENCAS twhile '(' EXP ')' ';'
{
    $$ = new DoWhile($2, $5, @1.first_line, @1.first_column);
}
| tdo BLOQUE_SENTENCAS twhile error EXP ')' BLOQUE_SENTENCAS { TablaError.insertarError(new Error("Sintactico", "El error en while falta '(' ",this._$.first_line, this._$.first_column)); }
| tdo BLOQUE_SENTENCAS twhile '(' EXP error BLOQUE_SENTENCAS { TablaError.insertarError(new Error("Sintactico", "El error en while falta ')'",this._$.first_line, this._$.first_column)); }
| tdo BLOQUE_SENTENCAS error '(' EXP ')' BLOQUE_SENTENCAS { TablaError.insertarError(new Error("Sintactico", "El error en while falta '(' ",this._$.first_line, this._$.first_column)); }
;

FOR : tfor '(' DECLARACION ':' EXP ':' ACTUALIZACION_FOR ')' BLOQUE_SENTENCAS
{
    $$ = new For($3, $5, $7, $9, @1.first_line, @1.first_column);
}
| tfor '(' ASIGNACION ':' EXP ':' ACTUALIZACION_FOR ')' BLOQUE_SENTENCAS
{
    $$ = new For($3, $5, $7, $9, @1.first_line, @1.first_column);
}
| tfor '(' error ':' EXP ':' ACTUALIZACION_FOR ')' BLOQUE_SENTENCAS { TablaError.insertarError(new Error("Sintactico", "El error en For falta declaracion o asignacion de una variable.",this._$.first_line, this._$.first_column)); }
| tfor '(' DECLARACION error EXP ':' ACTUALIZACION_FOR ')' BLOQUE_SENTENCAS { TablaError.insertarError(new Error("Sintactico", "El error en For falta ':' ",this._$.first_line, this._$.first_column)); }
| tfor '(' DECLARACION ':' EXP error ACTUALIZACION_FOR ')' BLOQUE_SENTENCAS { TablaError.insertarError(new Error("Sintactico", "El error en For falta ':' ",this._$.first_line, this._$.first_column)); }
| tfor '(' DECLARACION ':' EXP ':' error ')' BLOQUE_SENTENCAS { TablaError.insertarError(new Error("Sintactico", "El error en For falta el iterador ++/--/+n ",this._$.first_line, this._$.first_column)); }
| tfor '(' DECLARACION ':' EXP ':' ACTUALIZACION_FOR error BLOQUE_SENTENCAS { TablaError.insertarError(new Error("Sintactico", "El error en For falta ':' ",this._$.first_line, this._$.first_column)); }
;

WHILE : twhile '(' EXP ')' BLOQUE_SENTENCAS
{
    $$ = new While($3, $5, @1.first_line, @1.first_column);
}
| twhile error EXP ')' BLOQUE_SENTENCAS { TablaError.insertarError(new Error("Sintactico", "El error en while falta '(' ",this._$.first_line, this._$.first_column)); }
| twhile '(' EXP error BLOQUE_SENTENCAS { TablaError.insertarError(new Error("Sintactico", "El error en while falta ')'",this._$.first_line, this._$.first_column)); }
;

;
```

FUNCION: Contiene los diversos tipos de declaraci3n de una funci3n dentro del lenguaje.

```
FUNCION: TIPO id '(' LISTA_PARAM ')' BLOQUE_SENTENCAS
{
    $$ = new DeclararFuncion($1, $2, $4, $6, @2.first_line, @2.first_column);
}
| tvoid id '(' LISTA_PARAM ')' BLOQUE_SENTENCAS
{
    $$ = new DeclararFuncion(new Tipo(TipoPrimitivo.Void), $2, $4, $6, @2.first_line, @2.first_column);
}
| TIPO id '(' ')' BLOQUE_SENTENCAS
{
    $$ = new DeclararFuncion($1, $2, [], $5, @2.first_line, @2.first_column);
}
| tvoid id '(' ')' BLOQUE_SENTENCAS
{
    $$ = new DeclararFuncion(new Tipo(TipoPrimitivo.Void), $2, [], $5, @2.first_line, @2.first_column);
}
;
```

TIPOS: Hace referencia a los tipos primitivos.

```
TIPO : tinteger { $$ = new Tipo(TipoPrimitivo.Integer); }
| tboolean { $$ = new Tipo(TipoPrimitivo.Boolean); }
| tstring { $$ = new Tipo(TipoPrimitivo.String); }
| tdouble { $$ = new Tipo(TipoPrimitivo.Double); }
| tchar { $$ = new Tipo(TipoPrimitivo.Char); }
;
```

Listado para llamar parámetro en declaración o expresiones:

```
LISTA_PARAM : LISTA_PARAM ',' TIPO id
{
    let decla=new DeclararVariable($3, $4, undefined, @1.first_line, @1.first_column)
    $1.push(decla);
    $$ = $1;
}
| TIPO id
{
    let declal = new DeclararVariable($1, $2, undefined, @1.first_line, @1.first_column);
    let params = [];
    params.push(declal);
    $$ = params;
}
;

LISTA_EXP : LISTA_EXP ',' EXP
{
    $1.push($3);
    $$ = $1;
}
| EXP
{
    let lista_exp = [];
    lista_exp.push($1);
    $$ = lista_exp;
}
;
```

FUNCIONES DE LENGUAJE: Realizan el llamado de una función especial del lenguaje.

FUNCION LENGUAJE

```
: ttoLower '(' EXP ')' {$$ = new FuncionLenguaje($1, $3, @1.first_line, @1.first_column);}
| ttoUpper '(' EXP ')' {$$ = new FuncionLenguaje($1, $3, @1.first_line, @1.first_column);}
| ttruncate '(' EXP ')' {$$ = new FuncionLenguaje($1, $3, @1.first_line, @1.first_column);}
| tround '(' EXP ')' {$$ = new FuncionLenguaje($1, $3, @1.first_line, @1.first_column);}
| ttoCharArray '(' EXP ')' {$$ = new FuncionLenguaje($1, $3, @1.first_line, @1.first_column);}
| ttoString '(' EXP ')' {$$ = new FuncionLenguaje($1, $3, @1.first_line, @1.first_column);}
| ttypeof '(' EXP ')' {$$ = new FuncionLenguaje($1, $3, @1.first_line, @1.first_column);}
| tlength '(' EXP ')' {$$ = new FuncionLenguaje($1, $3, @1.first_line, @1.first_column);}
;
```

EXP: Contiene todas las expresiones permitidas dentro del lenguaje.

```
EXP : EXP '+' EXP { $$ = new OperacionAritmetica($1, $2, $3, @2.first_line, @2.first_column);}
| EXP '-' EXP { $$ = new OperacionAritmetica($1, $2, $3, @2.first_line, @2.first_column);}
| EXP '*' EXP { $$ = new OperacionAritmetica($1, $2, $3, @2.first_line, @2.first_column);}
| EXP '/' EXP { $$ = new OperacionAritmetica($1, $2, $3, @2.first_line, @2.first_column);}
| EXP '^' EXP { $$ = new OperacionAritmetica($1, $2, $3, @2.first_line, @2.first_column);}
| '-' EXP %prec negativo { $$ = new OperacionAritmetica($2, "negativo", $2, @2.first_line, @2.first_column);}
| '(' EXP ')' { $$ = $2;}
| EXP '%' EXP { $$ = new OperacionAritmetica($1, $2, $3, @2.first_line, @2.first_column);}
| EXP '==' EXP { $$ = new OperacionRelacional($1, $2, $3, @2.first_line, @2.first_column);}
| EXP '!=' EXP { $$ = new OperacionRelacional($1, $2, $3, @2.first_line, @2.first_column);}
| EXP '<' EXP { $$ = new OperacionRelacional($1, $2, $3, @2.first_line, @2.first_column);}
| EXP '>' EXP { $$ = new OperacionRelacional($1, $2, $3, @2.first_line, @2.first_column);}
| EXP '<=' EXP { $$ = new OperacionRelacional($1, $2, $3, @2.first_line, @2.first_column);}
| EXP '>=' EXP { $$ = new OperacionRelacional($1, $2, $3, @2.first_line, @2.first_column);}
| EXP '==' EXP { $$ = new OperacionRelacional($1, $2, $3, @2.first_line, @2.first_column);}
| EXP '==' EXP { $$ = new OperacionLogica($1, $2, $3, @2.first_line, @2.first_column);}
| EXP '||' EXP { $$ = new OperacionLogica($1, $2, $3, @2.first_line, @2.first_column);}
| id { $$ = new AccesoVariable($1, @1.first_line, @1.first_column);}
| id '[' EXP ']' { $$ = new AccesoVector($1, $3, @1.first_line, @1.first_column);}
| id '[' '[' EXP ']' ']' { $$ = new AccesoLista($1, $4, @1.first_line, @1.first_column);}
| LLAMADA_FUNCION { $$ = $1;}
| TERNARIA { $$ = $1;}
| CASTEO { $$ = $1;}
| FUNCION LENGUAJE { $$ = $1;}
| PRIMITIVO { $$ = $1;}
| error { TablaError.insertarError(new Error("Sintactico", `Error a nivel expresion despues de: "${yytext}" no cumplio o no entra`)); }
;
```