

Escuela de Ciencias y Sistemas

Organización de Lenguajes y Compiladores 1

Proyecto 1  
Manual Técnico

César André Ramírez Dávila

202010816

Fecha: 19/09/2022

## Tabla de contenido

<i>Introducción .....</i>	<b>3</b>
<i>Objetivos .....</i>	<b>3</b>
<i>Requerimientos .....</i>	<b>4</b>
<i>Creación del programa .....</i>	<b>5</b>
1. <b>Interfaz Gráfica .....</b>	<b>5</b>
2. <b>Carga de Archivo .....</b>	<b>5</b>
3. <b>Token de Errores .....</b>	<b>6</b>
4. <b>Analizador Lexico .....</b>	<b>6</b>
5. <b>Analizador Sintactico.....</b>	<b>10</b>
5.1.     Declaraciones .....	14
5.2.     Asignaciones .....	15
5.3.     IF .....	16
5.4.     Según .....	16
5.5.     Para.....	17
5.6.     Mientras .....	17
5.7.     Do While .....	17
5.8.     Retorno.....	17
5.9.     Método .....	17
5.10.    Funciones .....	18
5.11.    Ejecutar .....	18
5.12.    Imprimir .....	18
5.13.    ImprimirNL .....	18
6. <b>Gráfico del Árbol Sintáctico .....</b>	<b>19</b>

## Introducción

El presente documento describe los aspectos técnicos informáticos del programa. El documento familiariza al desarrollador que hace uso del lenguaje Java con ayuda de las herramientas Jflex y Jcup con temas como: uso de Clases, Gramática, Analizador léxico, Analizador Sintáctico, cargas de archivos, conversión de pseudocódigo a Lenguaje python o golang.

## Objetivos

- Instruir el uso adecuado del Sistema de información, para el acceso adecuado en el uso de este, mostrando los pasos de desarrollo del programa, así como la descripción de las funciones y métodos usados para la realizacion del programa.
- Comprender uso de las herramientas Jflex y Jcup en el lenguaje Java
- Obtener mayor conocimiento en el lenguaje Java

El presente manual está enfocado en el lenguaje de programación Java.

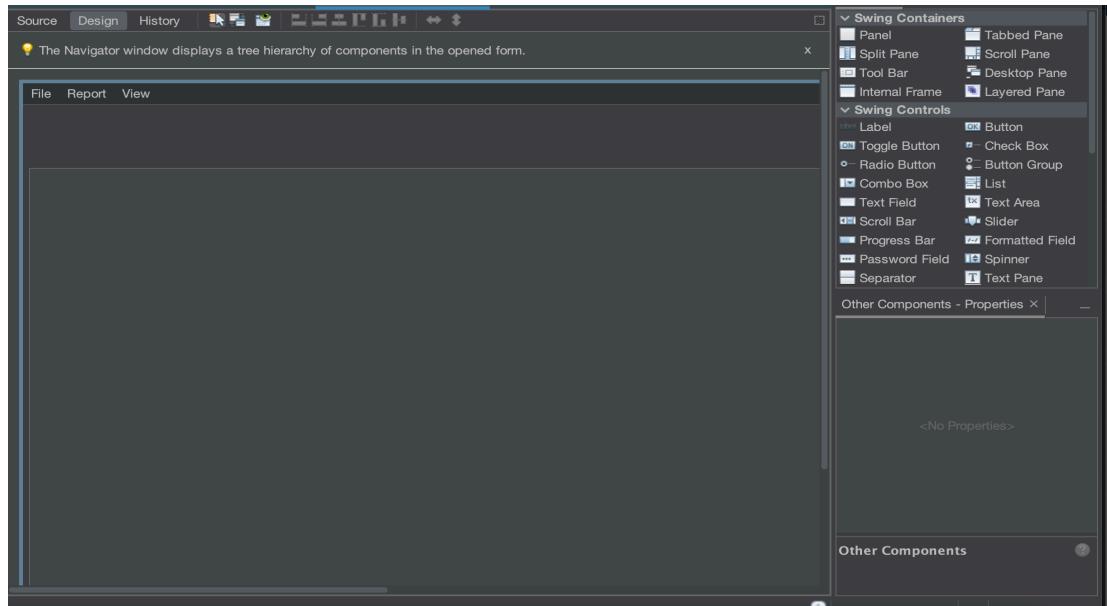
## Requerimientos

- La aplicación puede ser ejecutada en cualquier sistema operativo que tenga instalado Java en el sistema.
- IDE recomendado: Netbeans o visual studio code (opcional)
- Equipo Intel Pentium o superior
- Espacio en el disco duro, al menos 500 mb
- Memoria ram recomendada 2gb

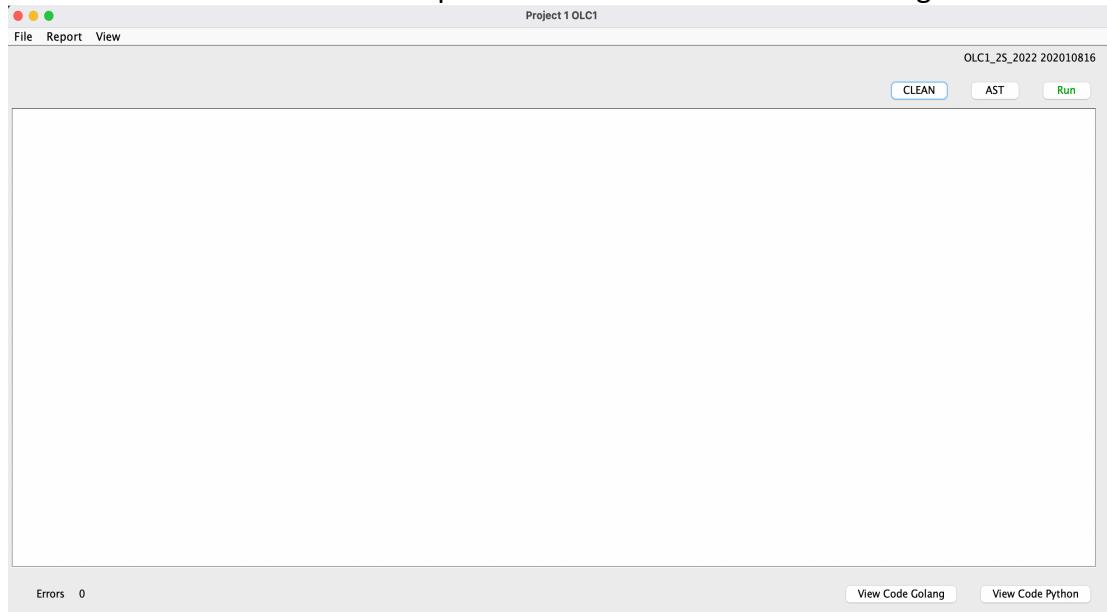
# Creación del programa

## 1. Interfaz Gráfica

La primera parte del programa es tener una interfaz grafica donde se procederá a la ejecución de todo el programa.



Con la herramienta de Netbeans para hacer interfacez diseñamos la siguiente



La cual dejamos la siguiente estructura.

## 2. Carga de Archivo

Creamos la función con nombre leerArchivos que nos permite hacer la carga de un archivo con extensión específica .olc y poner su contenido en la caja de texto para su edición posteriormente.

```
public void leerArchivos() {  
    try {  
        JFileChooser f = new JFileChooser();  
        f.setFileFilter(new FileNameExtensionFilter("File *.olc", "olc","OLC"));  
        int op = f.showOpenDialog(this);  
        if (op == JFileChooser.APPROVE_OPTION) {  
            System.out.println(f.getSelectedFile());  
            archivo = f.getSelectedFile();  
            JOptionPane.showMessageDialog(null,"Cargado con éxito!", "Carga Exitosa!", JOptionPane.INFORMATION_MESSAGE);  
        }  
        fr = new FileReader(archivo);  
        br = new BufferedReader(fr);  
        String linea;  
  
        while ((linea = br.readLine()) != null) {  
            contenido += linea;  
            cajatexto.append(linea + "\n");  
        }  
  
    } catch (Exception e) {  
        e.printStackTrace();  
    } finally {  
        try {  
            if (null != fr) {  
                fr.close();  
            }  
        } catch (Exception e2) {  
            e2.printStackTrace();  
        }  
    }  
}
```

### 3. Token de Errores

```
public class TError {  
  
    String tipo, lexema, descripcion;  
    int linea, columna;  
  
    public TError(String tipo, String lexema, String descripcion, int linea, int columna) {  
        this.tipo = tipo;  
        this.lexema = lexema;  
        this.descripcion = descripcion;  
        this.linea = linea + 1;  
        this.columna = columna + 1;  
    }  
  
    public String show() {  
        String data = "";  
        data += "\ntipo:" + tipo;  
        data += "\nlexema:" + lexema;  
        data += "\ndescripcion:" + descripcion;  
        data += "\nlinea:" + String.valueOf(linea);  
        data += "\ncolumna:" + String.valueOf(columna);  
        return data;  
    }  
  
    public String getLexema() {  
        return lexema;  
    }  
  
    public int getLine() {  
        return linea;  
    }  
  
    public int getColumn() {  
        return columna;  
    }  
}
```

Obtenemos su tipo, lexema, descripción, linea y columna

### 4. Analizador Léxico

El siguiente paso para el programa es la creación del Analizador léxico con la ayuda de Jflex

```
package analizadores;
import java_cup.runtime.*;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.LinkedList;
```

Importando las librerías que usaremos

```
%{
    public static LinkedList<TError> errores = new LinkedList<TError>();%}
```

Teniendo una clase de Token Error para ir almacenando cada que encuentre.

```
%public
%class Analizador_Lexico
%cupsym Simbolos
%cup
%char
%column
%full
%ignorecase
%line
%unicode
```

Haciendo más importaciones para su uso

```
letra = [a-zA-Z]
variable = ([\_] [^\n\_]* [\_])
cadena = [''][^\']*[\'']|[\""][^\"]*[\""]
caracter = ([\'][^\n\']*[\''])
numero = [0-9]+([.][0-9]+)?
comentario = ("//[^"\n"]+")
comentarioMulti = [\/][\*]+(((\s|\S)[\*\n]*))|([\n\*]*[^\n\*])|([\*]+[\n\*])
```

Expresiones regulares que se usan durante la ejecución del programa

```

%%%
<YYINITIAL>","      {
    //codigo en java
    System.out.println("Reconocio token: <coma> lexema: "+yytext());
    return new Symbol(Simbolos.tcoma, yycolumn, yyline, yytext());
}
<YYINITIAL>;"      {
    //codigo en java
    System.out.println("Reconocio token: <puntoycoma> lexema: "+yytext());
    return new Symbol(Simbolos.tpuntoycoma, yycolumn, yyline, yytext());
}
<YYINITIAL>+"      {
    //codigo en java
    System.out.println("Reconocio token: <tsuma> lexema: "+yytext());
    return new Symbol(Simbolos.tsuma, yycolumn, yyline, yytext());
}
<YYINITIAL>"-"      {
    //codigo en java
    System.out.println("Reconocio token: <guion> lexema: "+yytext());
    return new Symbol(Simbolos.tresta, yycolumn, yyline, yytext());
}
<YYINITIAL>*"      {
    //codigo en java
    System.out.println("Reconocio token: <asterisco> lexema: "+yytext());
    return new Symbol(Simbolos.tmultiplicacion, yycolumn, yyline, yytext());
}
<YYINITIAL>"/"      {
    //codigo en java
    System.out.println("Reconocio token: <diagonal> lexema: "+yytext());
    return new Symbol(Simbolos.tdivision, yycolumn, yyline, yytext());
}

```

### Empezando con los tokens que reconocerá nuestro programa

```

<YYINITIAL> "("   {
    //codigo en java
    System.out.println("Reconocio token: <parentesisAbre> lexema: "+yytext());
    return new Symbol(Simbolos.tparA, yycolumn, yyline, yytext());
}
<YYINITIAL> ")"  {
    //codigo en java
    System.out.println("Reconocio token: <parentesisCierra> lexema: "+yytext());
    return new Symbol(Simbolos.tparC, yycolumn, yyline, yytext());
}
<YYINITIAL> "["   {
    //codigo en java
    System.out.println("Reconocio token: <corcheteabre> lexema: "+yytext());
    return new Symbol(Simbolos.tcorA, yycolumn, yyline, yytext());
}
<YYINITIAL> "]"  {
    //codigo en java
    System.out.println("Reconocio token: <corchetecierra> lexema: "+yytext());
    return new Symbol(Simbolos.tcorC, yycolumn, yyline, yytext());
}
<YYINITIAL> "_"  {
    //codigo en java
    System.out.println("Reconocio token: <guion_bajo> lexema: "+yytext());
    return new Symbol(Simbolos.tguionBajo, yycolumn, yyline, yytext());
}
<YYINITIAL> "->" {
    //codigo en java
    System.out.println("Reconocio token: <flecha> lexema: "+yytext());
    return new Symbol(Simbolos.tflecha, yycolumn, yyline, yytext());
}

```

```

<YYINITIAL>"fin_metodo"  {
    //codigo en java
    System.out.println("Reconocio palabra_reservada, lexema: "+yytext());
    return new Symbol(Simbolos.prFinMetodo, yycolumn, yyline, yytext());
}

<YYINITIAL>"funcion"   {
    //codigo en java
    System.out.println("Reconocio palabra_reservada, lexema: "+yytext());
    return new Symbol(Simbolos.prFuncion, yycolumn, yyline, yytext());
}

<YYINITIAL>"fin_funcion"  {
    //codigo en java
    System.out.println("Reconocio palabra_reservada, lexema: "+yytext());
    return new Symbol(Simbolos.prFinFuncion, yycolumn, yyline, yytext());
}

<YYINITIAL>"ejecutar"   {
    //codigo en java
    System.out.println("Reconocio palabra_reservada, lexema: "+yytext());
    return new Symbol(Simbolos.prEjecutar, yycolumn, yyline, yytext());
}

<YYINITIAL>"imprimir"   {
    //codigo en java
    System.out.println("Reconocio palabra_reservada, lexema: "+yytext());
    return new Symbol(Simbolos.prImprimir, yycolumn, yyline, yytext());
}

<YYINITIAL>"imprimir_nl"   {
    //codigo en java
    System.out.println("Reconocio palabra_reservada, lexema: "+yytext());
    return new Symbol(Simbolos.prImprimirNl, yycolumn, yyline, yytext());
}

```

## Palabras reservadas que aceptará el programa

```

<YYINITIAL>{variable}  {
    System.out.println("Reconocio token: <variable> lexema: "+yytext());
    return new Symbol(Simbolos.tvariable, yycolumn, yyline, yytext());
}

<YYINITIAL> {numero} {
    System.out.println("Reconocio token: <numero> lexema: "+ yytext());
    return new Symbol(Simbolos.numero, yycolumn, yyline, yytext());
}

<YYINITIAL>{cadena} {
    System.out.println("Reconocio token: <cadena> lexema: "+ yytext());
    return new Symbol(Simbolos.tcadena, yycolumn, yyline, yytext());
}

<YYINITIAL>{caracter} {
    System.out.println("Reconocio token: <caracter> lexema: "+ yytext());
    return new Symbol(Simbolos.tcaracter, yycolumn, yyline, yytext());
}

<YYINITIAL>{comentario} {
    System.out.println("Reconocio token: <comentario> lexema: "+yytext());
}

<YYINITIAL>{comentarioMulti} {
    System.out.println("Reconocio token: <comentarioMulti> lexema: "+ yytext());
}

```

## Agregando las expresiones regulares definidas anteriormente

```

[ \t \n \f \r ] { /* Espacios en blanco se ignoran */

.

.

}

    System.out.println("Error Lexico : "+yytext()+" Linea "+(yyline+1)+" Columna "+(yycolumn+1));
    TError tmp= new TError("Lexico", yytext(),"NO PERTENECE AL LENGUAJE", yyline, yycolumn );
    errores.add(tmp);
}

```

Espacios y errores Léxicos que pueden venir

## 5. Analizador Sintactico

```

package analizadores;
import java_cup.runtime.Symbol;
import java.util.LinkedList;
import java.util.ArrayList;
import java.io.IOException;
import java.io.PrintWriter;

```

Empezamos importando las librerías que usaremos

```

parser code
{:
    public static Nodo1 padre;
    public static Nodo2 padre1;
    public int cont = 0;
    public int cont2 = 0;
    public static LinkedList<TError> errores = new LinkedList<TError>();
    public static ArrayList<String> vars = new ArrayList<String>();

    public static String tipo="Lista";
    public String codigoTraducidoPython="";
    public String pruebas = "";
    public String codigoTraducidoGolang="";
    public String errorm="";
}

```

Usamos la siguiente instrucción de parser y dentro de ella podemos poner código de Java.

```

/**
 * Método al que se llama automáticamente ante algún error sintáctico.
 */
public void syntax_error(Symbol s){
    String lexema = s.value.toString();
    int fila = s.right;
    int columna = s.left;

    System.out.println("Error Sintáctico en la Línea " + (s.right + 1) +
        " Columna "+s.left+". No se esperaba este carácter: " +s.value+".");
    TError tmp = new TError("Sintáctico",lexema,"Caracter no esperado",fila,columna);
    errores.add(tmp);
}

```

Método para que el programa se recupere de algún error sintáctico

```

/**
 * Método al que se llama automáticamente ante algún error sintáctico
 * en el que ya no es posible una recuperación de errores.
 */
public void unrecovered_syntax_error(Symbol s) throws java.lang.Exception{
    String lexema = s.value.toString();
    int fila = s.right;
    int columna = s.left;

    System.out.println("Error sintáctico irrecuperable en la Línea " +
        (s.left + 1)+ " Columna "+s.right+". Componente " + s.value +
        " no reconocido.");

    TError tmp = new TError("Sintáctico",lexema, "TOKEN NO ERA EL ESPERADO",fila,columna);
    errores.add(tmp);
}

```

Método donde ya no es posible recuperarse de algún error sintáctico

```

//definicion de terminales
terminal String numero;
terminal String tvariable;
terminal String tcaracter;
terminal String tpunto;
terminal String tcoma;
terminal String tpuntoycoma;
terminal String tcadena;
terminal String tsuma;

```

Empezamos con la definición de los terminales para la gramática

```

terminal String prRetornar;
terminal String prMetodo;
terminal String prConParametros;
terminal String prFinMetodo;
terminal String prFuncion;
terminal String prFinFuncion;
terminal String prEjecutar;
terminal String prImprimir;
terminal String prImprimirNl;
terminal String prFin;

```

Así con todos los tokens que se definieron en el analizador léxico

```

//definicion de no terminales
non terminal INICIO,INSTRUCCIONES,INSTRUCCION;
non terminal DECLARACIONES,ASIGNACIONES,IF,LISTAOSI,SEGUN,CICLO_FOR, CICLO WHILE;
non terminal CICLO_DOWHILE,RETORNO,METODO,FUNCIONES,EJECUTAR,IMPRIMIR,IMPRIMIRNL;
non terminal DECLARARNUML,DECLARARCADENA,DECLARARCADENAL,DECLARARBOOLEANO,DECLARARBOOLEANOL,DECLARARCARACTER;
non terminal DECLARARCARACTERL,LISTAVARIABLES,LISTAVARIABLES2,DECLARARCONOPERACION,CONDICIONIF;
non terminal ASIGNARNUM,ASIGNARNUML,ASIGNARCADENA,ASIGNACADENAL,ASIGNARBOOLEANO,ASIGNARBOOLEANOL,ASIGNARCARACTER,ASIGNARCAR;
non terminal EXPRESIONARITMETICA,EXPRESIONES,EXPRESION, INSTRUCCIONSEGUN;
non terminal CONDICIONMIENTRAS, LISTAPARAMETROS, TIPODATO;
non terminal RELACIONAL,LOGICO;

```

Definimos los no terminales

```

//gramaticas
start with INICIO;

INICIO ::= prInicio:pinic INSTRUCCIONES:n prFin:pfin {:

```

Empezamos con la Gramática del programa

```

Nodo1 nd = new Nodo1();
Nodo1 nd1 = new Nodo1();
Nodo1 nd2 = new Nodo1();
Nodo1 nd3 = new Nodo1();
Nodo1 nd4 = new Nodo1();
nd.setEtiqueta("GLOBAL");
nd.setIdNod(parser.cont);
nd1.setEtiqueta("prInicio");
nd1.setValor(pinic.toString());
nd2.setEtiqueta("prFin");
nd3.setEtiqueta(pinic.toString());
nd4.setEtiqueta(pfin.toString());
nd1.setIdNod(parser.cont);
nd2.setIdNod(parser.cont);
parser.cont++;
nd.AddHijos ((Nodo1) nd1);
nd.AddHijos((Nodo1) n);
nd.AddHijos((Nodo1) nd2);
nd1.AddHijos((Nodo1) nd3);
nd2.AddHijos((Nodo1) nd4);
parser.padre = (Nodo1) nd;
RESULT= nd;

```

Método para la realización del Árbol Sintáctico

```
INSTRUCCIONES ::= INSTRUCCION:n  INSTRUCCIONES:n1 { : RESULT=n;  
RESULT=n1;  
Nodo1 nd = new Nodo1();  
nd.setEtiqueta("Instrucciones");  
nd.setIdNod(parser.cont);  
parser.cont++;  
nd.AddHijos((Nodo1) n);  
nd.AddHijos((Nodo1) n1);  
RESULT= nd;  
:}
```

Seguimos con las instrucciones, puede venir una única instrucción o puede venir n cantidad de instrucciones

En INSTRUCCIÓN tenemos los siguientes no terminales

- Declaraciones
- Asignaciones
- Ciclo IF
- Ciclo Según
- Ciclo For
- Ciclo While
- Ciclo Do While
- Retorno
- Metodo
- Funciones
- Ejecutar
- Imprimir
- ImprimirNI

```
INSTRUCCION ::= DECLARACIONES:n { : RESULT=n;  
Nodo1 nd = new Nodo1();  
nd.setEtiqueta("Declaracion");  
nd.setIdNod(parser.cont);  
parser.cont++;  
nd.AddHijos((Nodo1) n);  
RESULT= nd;  
:}
```

Siguiendo toda la siguiente estructura

```

        | EJECUTAR:n {: RESULT=n;
Nodo1 nd = new Nodo1();
nd.setEtiqueta("Ejecutar");
nd.setIdNod(parser.cont);
parser.cont++;
nd.AddHijos((Nodo1) n);
RESULT= nd;
:}
        | IMPRIMIR:n {: RESULT=n;
Nodo1 nd = new Nodo1();
nd.setEtiqueta("Imprimir");
nd.setIdNod(parser.cont);
parser.cont++;
nd.AddHijos((Nodo1) n);
RESULT= nd;
:}
        | IMPRIMIRNL:n1 {: RESULT=n1;
Nodo1 nd = new Nodo1();
nd.setEtiqueta("Imprimir_nl");
nd.setIdNod(parser.cont);
parser.cont++;
nd.AddHijos((Nodo1) n1);
RESULT= nd;
:}
        | error:err {: RESULT = err;:}
;

```

Por último, definimos que puede venir un error o n errores

### 5.1. Declaraciones

```

//gramatica para las declaraciones

DECLARACIONES::= DECLARARCONOPERACION:n {: RESULT= n; :}
                | DECLARARNUML:n {:RESULT= n; :}
                | DECLARARCADENA:n {:RESULT= n; :}
                | DECLARARCADENAL:n {: RESULT= n; :}
                | DECLARARBOOLEANO:n {: RESULT= n;:}
                | DECLARARBOOLEANOL:n {: RESULT= n; :}
                | DECLARARCARACTER:n {: RESULT= n; :}
                | DECLARARCARACTERL:n {: RESULT= n; :}
;

```

Seguimos con las declaraciones

```

DECLARARCONOPERACION::= prIngresar:pri tvariable:a prComo:co prNumero:nm prConValor:c1 EXPRESIONARITMETICA:b tpuntoycoma:com
;
```

```

EXPRESIONARITMETICA::= EXPRESIONARITMETICA:b EXPRESIONES:a {: RESULT=b;:}
                    | EXPRESIONES:a {: RESULT=a;:}
;

EXPRESIONES::= EXPRESION:e {: System.out.println("= " + e + "="); RESULT=e; :}
                | RELACIONAL:e {: RESULT= e; :}
                | LOGICO:e {: RESULT = e;:}
;

```

```

EXPRESION ::= numero:n { : RESULT=n; :}
| EXPRESION:I tsuma EXPRESION:r { : RESULT= I + " " + r ; :}
| EXPRESION:I tresta EXPRESION:r { : RESULT= I + " - " + r ; :}
| EXPRESION:I tmultiplicacion EXPRESION:r { : RESULT= I + " * " + r ; :}
| EXPRESION:I tdivision EXPRESION:r { : RESULT= I + " / " + r ; :}
| EXPRESION:I prMod EXPRESION:r { : RESULT= I + "% " + r ; :}
| EXPRESION:I prPotencia EXPRESION:r { : RESULT= I + "%%" + r ; :}
| tparA:para EXPRESION:e tparC { : RESULT= "(" + e + ")"; :}
| tc当地:para EXPRESION:e tc当地C { : RESULT= "[" + e + "]"; :}
| tparA:para RELACIONAL:e tparC { : RESULT= "(" + e + ")"; :}
| tc当地A:para RELACIONAL:e tc当地C { : RESULT= "[" + e + "]"; :}
| tparA:para LOGICO:e tparC { : RESULT= "(" + e + ")"; :}
| tc当地A:para LOGICO:e tc当地C { : RESULT= "[" + e + "]"; :}
| tvariable:b { : RESULT= b; :}

;

RELACIONAL ::= EXPRESION:op1 prMayor:signo EXPRESION:op2 { : RESULT=op1+">" + op2; :}
| EXPRESION:op1 prMenor:signo EXPRESION:op2 { : RESULT=op1+"<" + op2; :}
| EXPRESION:op1 prMayorIgual:signo EXPRESION:op2 { : RESULT=op1+">=" + op2; :}
| EXPRESION:op1 prMenorIgual:signo EXPRESION:op2 { : RESULT=op1+"<=" + op2; :}
| EXPRESION:op1 prEsIGUAL:signo EXPRESION:op2 { : RESULT=op1+"==" + op2; :}
| EXPRESION:op1 prEsDiferente:signo EXPRESION:op2 { : RESULT=op1+"!=" + op2; :}

;

LOGICO ::= EXPRESION:op1 prOR:signo EXPRESION:op2 { : RESULT=op1+"or" + op2; :}
| EXPRESION:op1 prAND:signo EXPRESION:op2 { : RESULT=op1+"and" + op2; :}
| EXPRESION:op1 prNOT:signo EXPRESION:op2 { : RESULT=op1+"not" + op2; :}
;

```

Definiendo más instrucciones para el lenguaje

```

DECLARARNUML ::= prIngresar:pri LISTAVARIABLES:a prComo:co prNumero:nm prConValor:c1 numero:b tpuntoycoma:comaa {:
    f = f.replace("_", " ");
}

DECLARARCADENA ::= prIngresar:pri tvariable:a prComo:co prCadena:cad prConValor:c1 tcadena:b tpuntoycoma:comaa {:
    f = f.replace("_", " ");
}

```

Siguiendo con las demás declaraciones

## 5.2. Asignaciones

```

// gramatica para las asignaciones

ASIGNACIONES ::= ASIGNARNUM:n { : RESULT=n; :}
| ASIGNARNUML:n { : RESULT=n; :}
| ASIGNARCADENA:n { : RESULT=n; :}
| ASIGNARCADENAL:n { : RESULT=n; :}
| ASIGNARBOOLEANO:n { : RESULT=n; :}
| ASIGNARBOOLEANOL:n { : RESULT=n; :}
| ASIGNARCARACTER:n { : RESULT=n; :}
| ASIGNARCARACTERL:n { : RESULT=n; :}
;

```

Empezamos las asignaciones, algo parecido a las declaraciones

```

ASIGNARNUM ::= tvariable:a tflecha:flech EXPRESIONARITMETICA:b tpuntoycoma:comaa { : a = a.replace("_", ""); }

```

```
ASIGNARCADENA ::= tvariable:a tflecha:flech tcadena:b tpuntoycoma:comaa {:
    a = a.replace("_", "");
```

Continuando con las demás asignaciones

```
ASIGNARCARACTER ::= tvariable:a tflecha:flech tcaracter:b tpuntoycoma:comaa {:
    a = a.replace("_", "");
```

### 5.3. IF

```
//Sentencias
// IF

IF ::= prSi:iif CONDICIONIF:a prEntonces:en INSTRUCCIONES:in prFinSi:fnsi {:
    RESULT += "if" + a + "\n" + in;
```

```
LISTAOSI ::= prOSi:osii CONDICIONIF:a prEntonces:en INSTRUCCIONES:in LISTAOSI:lis;
```

En el if pueden venir las instrucciones “0\_si” n cantidad de veces, para ello hacemos una lista.

```
CONDICIONIF ::= tvariable:a prEsIGUAL EXPRESIONARITMETICA:b {:
    a = a.replace("_", "");}
    RESULT = "(" + a + "==" + b + ")" + "\n"; :}
    | tvariable:a {:
        a = a.replace("_", "");}
    RESULT = "(" + a + ")" + "\n"; :}
    | tvariable:a prMayorIgual EXPRESIONARITMETICA:b {:
        a = a.replace("_", "");}
    RESULT = "(" + a + ">=" + b + ")" + "\n"; :}
    | tvariable:a prMenorIgual EXPRESIONARITMETICA:b {:
        a = a.replace("_", "");}
    RESULT = "(" + a + "<=" + b + ")" + "\n"; :}
    | tvariable:a prMayor EXPRESIONARITMETICA:b {:
        a = a.replace("_", "");}
    RESULT = "(" + a + ">" + b + ")" + "\n"; :}
    | tvariable:a prMenor EXPRESIONARITMETICA:b {:
        a = a.replace("_", "");}
    RESULT = "(" + a + "<" + b + ")" + "\n"; :}
    | tvariable:a prOR EXPRESIONARITMETICA:b {:
        a = a.replace("_", "");}
    RESULT = "(" + a + "or" + b + ")" + "\n"; :}
    | tvariable:a prAND EXPRESIONARITMETICA:b {:
        a = a.replace("_", "");}
    RESULT = "(" + a + "and" + b + ")" + "\n"; :}
    | prNOT EXPRESIONARITMETICA:b {:
        a = a.replace("_", "");}
    RESULT = "(" + "not" + b + ")" + "\n"; :}
;
```

Condiciones que puede tener el IF

### 5.4. Según

```
SEGUN ::= prSegun:seg EXPRESIONARITMETICA:exp prHacer:hac INSTRUCCIONSEGUN:inseg prDeloContrario:contr prEntonces:en INSTRUCCIONSEGUN:ins;
```

```
INSTRUCCIONSEGUN ::= tinterrogaciona:interA EXPRESIONARITMETICA:exp tinterrogacionc:interC prEntonces:en INSTRUCCIONES:in;
```

Instrucciones que pueden venir en el según

```
Nodo1 nd = new Nodo1();
Nodo1 nd1 = new Nodo1();
Nodo1 nd2 = new Nodo1();
Nodo1 nd3 = new Nodo1();
Nodo1 nd4 = new Nodo1();
nd.setEtiqueta("Instrucción Según");
```

## 5.5. Para

```
CICLO_FOR ::= prPara:par tvariable:a tflecha:flech EXPRESIONARITMETICA:b prHasta:hast EXPRESIONARITMETICA:c prHacer:hac INST
```

```
INSTRUCCIONES:d prFinPara:fnpara {: a = a.replace("_","");
RESULT=d;
```

## 5.6. Mientras

```
CICLO WHILE ::= prMientras:mien CONDICIONMIENTRAS:con prHacer:hac INSTRUCCIONES:in prFinMientras:fnmientras
```

```
CONDICIONMIENTRAS ::= tvariable:a prEsIGUAL EXPRESIONARITMETICA:b {: a = a.replace("_", "");  
RESULT = "(" + a + "==" + b + ")" + "\n"; :}  
| tvariable:a prMayorIgual EXPRESIONARITMETICA:b {: a = a.replace("_", "");  
RESULT = "(" + a + ">=" + b + ")" + "\n"; :}  
| tvariable:a {: a = a.replace("_", "");  
RESULT = "while" + "(" + a + ")" + "\n"; :}  
| tvariable:a prMenorIgual EXPRESIONARITMETICA:b {: a = a.replace("_", "");  
RESULT = "while" + "(" + a + "<=" + b + ")" + "\n"; :}  
| tvariable:a prMayor EXPRESIONARITMETICA:b {: a = a.replace("_", "");  
RESULT = "while" + "(" + a + ">" + b + ")" + "\n"; :}  
| tvariable:a prMenor EXPRESIONARITMETICA:b {: a = a.replace("_", "");  
RESULT = "while" + "(" + a + "<" + b + ")" + "\n"; :}  
| tvariable:a prOR numero:b {: a = a.replace("_", "");  
RESULT = "while" + "(" + a + "or" + b + ")" + "\n"; :}  
| tvariable:a prAND numero:b {: a = a.replace("_", "");  
RESULT = "while" + "(" + a + "and" + b + ")" + "\n"; :}  
| prNOT EXPRESIONARITMETICA:b {:  
RESULT = "while" + "(" + "not" + b + ")" + "\n"; :}  
;
```

Condiciones que pueden venir en el Ciclo Mientras

## 5.7. Do While

```
CICLO_DOWHILE ::= prRepetir:rep INSTRUCCIONES:in prHastaQue:hastq CONDICIONMIENTRAS:con {:  
Nodo1 nd = new Nodo1();  
Nodo1 nd1 = new Nodo1();
```

## 5.8. Retorno

```
RETORNO ::= prRetornar:retrn EXPRESIONARITMETICA:b tpuntoycoma:comaa {: RESULT += "\treturn " + b.toString() + "\n";  
codigoTraducidoPython += RESULT;  
Nodo1 nd = new Nodo1();  
Nodo1 nd1 = new Nodo1();
```

## 5.9. Método

```
METODO ::= prMetodo:met tvariable:b INSTRUCCIONES:in prFinMetodo:fnmetodo {:  
pruebas= "\tdef " + b + "():\n" + "\n" + in + "\n";  
codigoTraducidoPython += pruebas;  
Nodo1 nd = new Nodo1();
```

```

LISTAPARAMETROS ::= tvariable:a prNumero:b {:: a = a.replace("_", ""); RESULT= a + b; ::}
| tvariable:a prCadena:b {:: a = a.replace("_", ""); RESULT= a + b; ::}
| tvariable:a prCaracter:b {:: a = a.replace("_", ""); RESULT= a + b; ::}
| tvariable:a prBoolean:b {:: a = a.replace("_", ""); RESULT= a + b; ::}
| LISTAPARAMETROS:a tcoma:b tvariable:c prNumero:d {::c = c.replace("_", ""); RESULT= a + b + c + d; ::}
| LISTAPARAMETROS:a tcoma:b tvariable:c prCadena:d {::c.replace("_", ""); RESULT= a + b + c + d; ::}
| LISTAPARAMETROS:a tcoma:b tvariable:c prCaracter:d {:: c.replace("_", ""); RESULT= a + b + c + d; ::}
| LISTAPARAMETROS:a tcoma:b tvariable:c prBoolean:d {:: c.replace("_", ""); RESULT= a + b + c + d; ::}
;

```

Lista de Parámetros que pueden venir

### 5.10. Funciones

```

FUNCIONES ::= prFuncion:func tvariable:a TIPODATO:tdata INSTRUCCIONES:in prFinFuncion:fncfuncion
RESULT= "def " + a + "():\n" + in + "\n" + "return\n";
Nodo1 nd = new Nodo1();
Nodo1 nd1 = new Nodo1();

```

```

TIPODATO ::= prNumero:b {:: RESULT= b; ::}
| prCadena:b {:: RESULT= b; ::}
| prCaracter:b {:: RESULT= b; ::}
| prBoolean:b {:: RESULT= b; ::}
;

```

Tipo de dato que puede venir

### 5.11. Ejecutar

```

EJECUTAR ::= prEjecutar:ejec tvariable:a tparA:parA tparC:parC tpuntoycoma:comaa
codigoTraducidoPython += RESULT;
Nodo1 nd = new Nodo1();

```

### 5.12. Imprimir

```

IMPRIMIR ::= prImprimir:pri tcadena:n tpuntoycoma:comaa
//codigoTraducidoPython += "print" + "(" + n + ")" + "

```

### 5.13. ImprimirNL

```

IMPRIMIRNL ::= prImprimirNL:pri tcadena:n tpuntoycoma:comaa {
//codigoTraducidoPython += "print" + "(" + n + ")" + "\n";
//codigoTraducidoPython += RESULT;
}

```

## 6. Gráfico del Árbol Sintáctico

```
public static String recorrido(Nodo1 raiz){
    String cuerpo = "";
    for(Nodo1 hijos : raiz.hijos){
        if (!(hijos.Etiqueta.equals("Vacio"))){
            cuerpo += "\\" + raiz.idNod + "." + raiz.Etiqueta + "=" + raiz.valor + "\\->" + hijos.idNod + "," + hijos.Etiqu
        } else{
            System.out.println(raiz.Etiqueta);
        }
    }
    return cuerpo;
}
```

Empezamos el recorrido de los nodos y vamos agregándolo al contenido

```
public static void Graficar(String cadena, String cad) throws IOException {
    FileWriter fichero = null;
    PrintWriter pw = null;
    String nombre = cad;
    String archivo = nombre + ".dot";

    try{
        fichero = new FileWriter(archivo);
        pw = new PrintWriter(fichero);
        pw.println("digraph G {node[shape=ellipse, style=filled, color=darkgreen]\"");
        pw.println(cadena);
        pw.println("\n}");
        fichero.close();

    } catch (Exception e){
        System.out.println(e);
    }
}
```

Luego lo pasamos al formato de Graphviz.

### LINK REPOSITORIO

<https://github.com/AndreRD1026/OLC1-202010816>