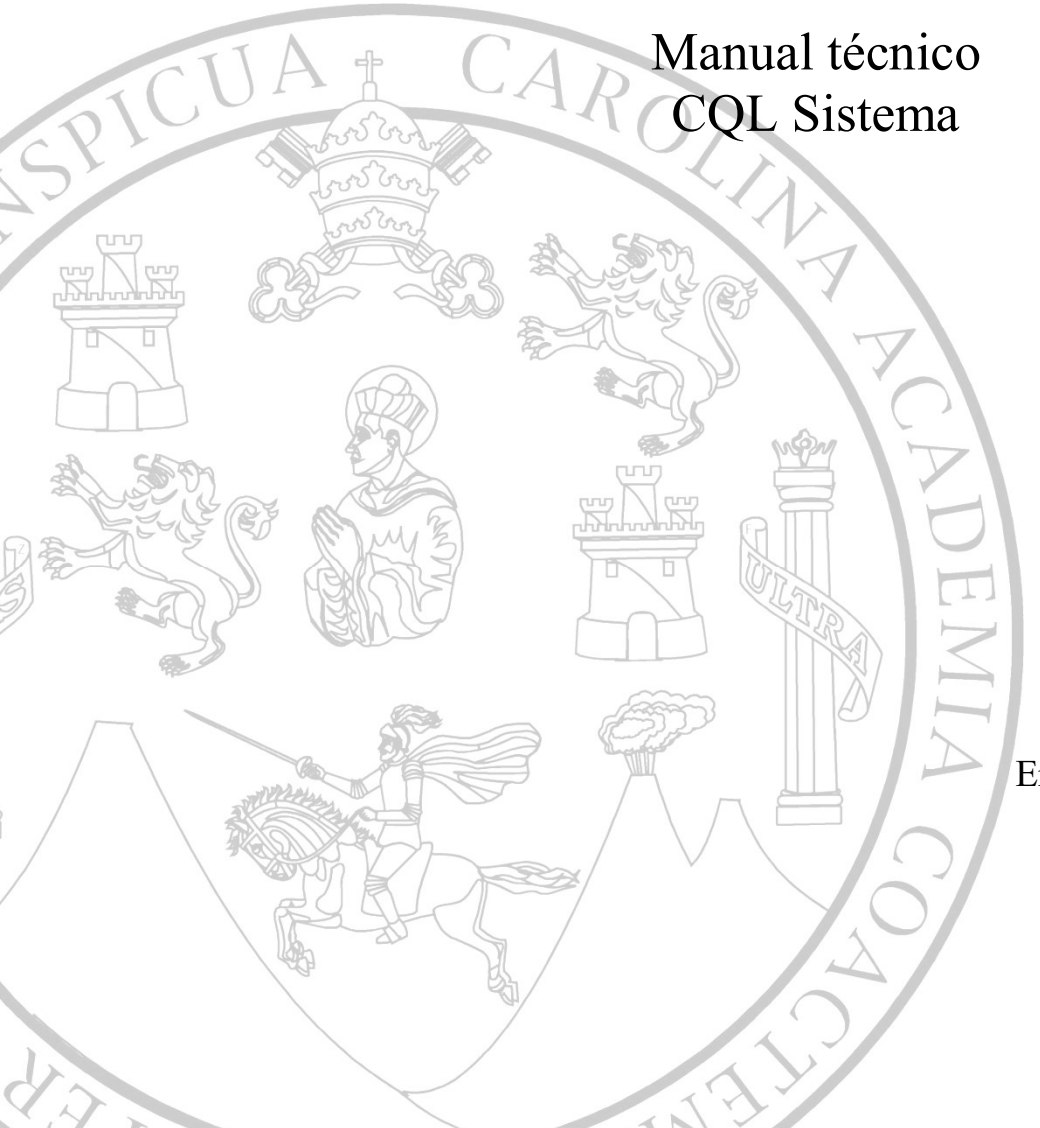
Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de ciencias y sistemas
Organización de lenguajes y compiladores 2
Sección A



Manual técnico
CQL Sistema

Erick Tejaxún 201213050

Requisitos del sistema

- Sistema operativo Windows 10 32 bits o 64 bits. / Sistema operativo GNU/Linux 32 o 64 bits.
- Mínimos 2GB de memoria RAM.
- Mínimo Procesador core i3.
- java versión 11.8.0_201
- Java(TM) SE Runtime Environment (build 1.8.0_201-b09)
- Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode)
- Netbeans 8.1
- JFLex 1.7.full
- CUP 1.0
- RSyntax

Diccionario de Paquetes

Nombre	Descripción
Analisis.FS	Paquete que contiene las clases necesarias para realizar el análisis sintáctico y análisis léxico sobre el lengauje FS.
Analisis.FS.AST	Paquete que contiene todas las clases que representan cada una de las instrucciones del lenguaje FS.
Analisis.GDato	Paquete que contiene las clases necarias para realizar el análisis léxico, sintáctico y semántico para los archivos gdato.
Analisis.XML	Paquete que contiene las clases necesarias para realizar el análisis sintáctico y análisis léxico sobre el lengauje GMXL.
Analisis.XML.AST	Paquete que contiene todas las clases que representan cada una de las instrucciones del lenguaje GXML.
CreatorXML201213050	Paquete que contiene la clases principal del sistema y también la ventana principal.
Recursos	Paquete que contiene los recursos necesarios para manejo de errores, y también otras estructuras necesarias para el funcionamiento del sistema como el Display.

Diccionario de clases paquete Analisis.FS

Nombre	Descripción
ParserFS	Analizador sintáctico generado a través de CUP
ScannerFS	Analizador léxico generado a través de JFlex
Sym	Diccionario de no terminales generados por CUP

Diccionario de clases paquete Analisis.FS.AST

Nombre	Descripción
Acceso	Instrucción de acceso a atributos de objetos.
AccesoArray	Instrucción de acceso a arreglos.
And	Instrucción lógica and
Asignación	Instrucción de asignación de valores a variables.
Aumento	Instrucción de aumento ++.
Bloque	Lista de instrucciones para cada bloque.
BoolExp	Instrucción que devuelve y almacena un valor booleano.
Caso	Lista de instrucciones en cada uno de los casos para la selección.
Declaración	Instrucción para la declaración de variables.
Decremento	Instrucción para decrementar valores --.
Entorno	Clase que permite simular un entorno y contiene nuestra tabla de símbolos.
Diferente	Instrucción relacional.
Div	Operación aritmética división.
DoubleExp	Instrucción que almacena un valor doublé.
IF	Instrucción para la evaluación IF.
Menor	Expresión relacional.

Mayor	Expresión relacional
Menorque	Expresión relacional.
Menorque	Expresion relacional.
Metodo	Instrucción método para almacenar el id, los atributos y el bloque de instrucciones.
Nativa	Instrucciones nativas para la generación de interfaz y manejo de arreglos.
Nodo	Clase abstracta para poder implementar el patrón de diseño interpreter.

Diccionario de clases paquete Analisis.XML

Nombre	Descripción
Parserxml	Analizador sintáctico generado a través de CUP
Scannerxml	Analizador léxico generado a través de JFlex
Sym	Diccionario de no terminales generados por CUP

Diccionario de clases paquete Analisis.XML.AST

Nombre	Descripción
Botón	Clase para la generación del objeto visual botón y enviar.
Contenedor	Clase para la generación de un contendor de objetos visuales.
Controlador	Clase para la generación del objeto visual controlador.
Dato	Clase que almacena datos para un controlador.
Defecto	Clase para almacenar el valor defecto de un controlador.
Importar	Instrucción importar archivos fs y gmxl.
Multimedia	Clase para la generación del objeto visual tipo multimedia.

Texto	Clase para la generación del objeto visual tipo texto.
Ventana	Clase para la generación del objeto visual tipo ventana.

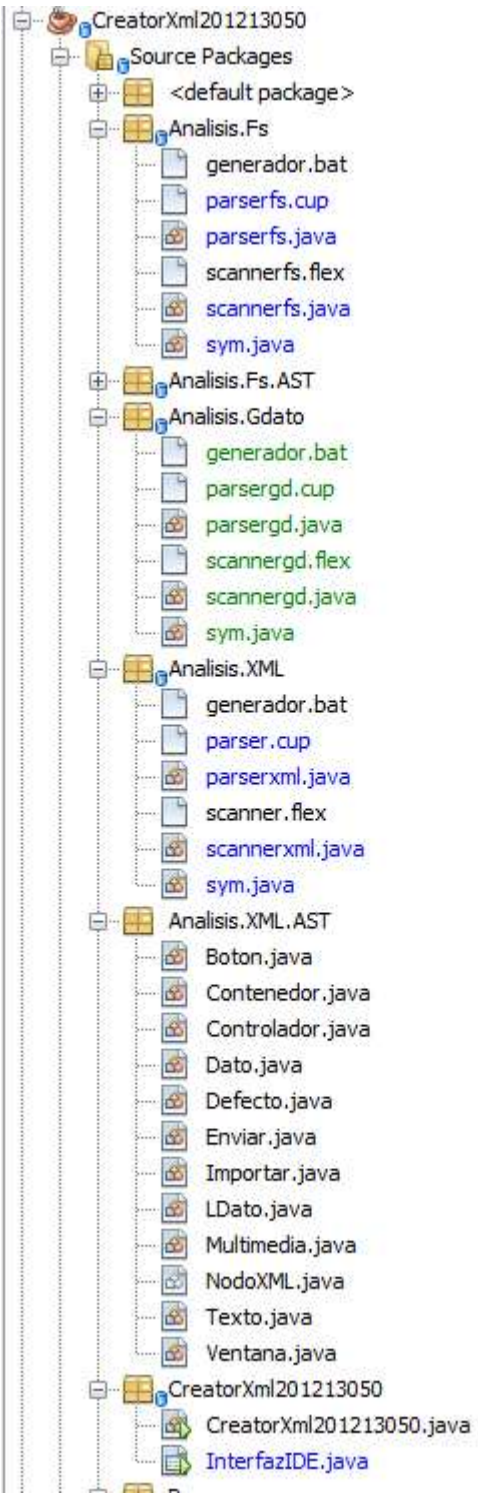
Diccionario de clases paquete Recursos

Nombre	Descripción
Display	Estructura que nos ayuda para poder ejecutar el código de alto nivel FS.
Error	Clase de error.
Lexema	Clase lexema léxico.
Singlenton	Clase que nos permite guardar los errores en cualquier lado, además de los retornos de métodos.

Diccionario de clases paquete CreatorXML201213050

Nombre	Descripción
Ventana	Interfaz del sistema
CreatorXML201213050	Clase principal del sistema

Esquema de paquetes



```
package Analisis;
import java_cup.runtime.Symbol;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import AST.*;
import Utilidades.ErrorC;
import AST.Instruccion.*;
import AST.Instruccion.Ciclos.*;
import AST.Expresion.*;
import AST.Entorno.*;
import AST.Expresion.Aritmetica.*;
import AST.Expresion.Relacional.*;
import AST.Expresion.Logica.*;
import AST.Expresion.Arreglo.*;
import AST.Expresion.Casteo.*;
import AST.Expresion.Casteo.Explicito.*;
import AST.Expresion.Funcion.*;
import AST.Clase.*;


parser code
{:
    public ArrayList<ErrorC> listaErrores = new ArrayList();
    public ArrayList<Instruccion> ast = new ArrayList<Instruccion>();

    public AST raiz = null;

    /*@Override
    public void syntax_error(Symbol cur_token)
    {
        List<Integer> listaIdTokens = expected_token_ids();
        LinkedList<String> listaNombres = new LinkedList<String>();
        for (Integer expected : listaIdTokens)
        {
            listaNombres.add(symbl_name_from_id(expected));
        }
        Utilidades.Singlenton.registrarError(String.valueOf(cur_token
.value),
```

```
            String.valueOf(cur_token.value) + ". Se esperaba :"+lista
Nombres.toString(), ErrorC.TipoError.LEXICO, cur_token.right+1, cur_t
oken.left+1);
        /*listaErrores.add(
                    listaErrores.add(new ErrorC(ErrorC.TipoError.SINT
ACTICO,
                                String.valueOf(cur_token.value) + ".
Se esperaba :"+listaNombres.toString(),
                                cur_token.right+1,
                                cur_token.left+1));

    }*/


    public void report_error(String message, Object info)
    {
        int linea = 0;
        int columna = 0;
        java_cup.runtime.Symbol s = null;
        StringBuilder m = new StringBuilder("Error Sintactico");

        if (info instanceof java_cup.runtime.Symbol)
        {
            s = ((java_cup.runtime.Symbol) info);
            if (s.left >= 0)
            {
                columna = s.left+1;
                if (s.right>= 0)
                {
                    linea = s.right+ 1;
                }
            }
        }

        m.append(" Se esperaba: "+message);
        //System.err.println(m.toString());
        //System.out.println("Error");
        //System.out.println("Error linea:"+linea+", col:"+columna);
        LinkedList<String> toks = new LinkedList();

        if(!expected_token_ids().isEmpty())
        {
```

```
            Imprimir("No esta vacia "+ expected_token_ids().size());

            for(int w=0; w<expected_token_ids().size(); w++)
            {
                if(expected_token_ids().get(w) !=sym.error)
                {
                    int tok = (int)expected_token_ids().get(w);

                    toks.add( symbol_name_from_id(tok) );
                }
            }
        }

        Imprimir(expected_token_ids().size());

        for(int w=0; w<expected_token_ids().size(); w++)
        {
            if(expected_token_ids().get(w)!=sym.error)
            {
                int tok = (int)expected_token_ids().get(w);
                toks.add( symbol_name_from_id(tok) );
            }
        }
        String esperados = "";
        for(String id : toks)
        {
            if(!esperados.equals(""))
            {
                esperados += ", ";
            }
            esperados += id;

        }
        Utilidades.Singlenton.registrarError(s.value.toString(), "Se
esperaba .. " +esperados, ErrorC.TipoError.SINTACTICO, linea, columna
);
    }

    public String symbol_name_from_id(int id){
        return sym.terminalNames[id];
    }

    /*public void addError(Symbol s)
```

```
    {
        listaErrores.add(new ErrorC("Sintactico",s.rights.right+1,Con
vertirObjectToString(s.value)));
    }*/
    public void Imprimir(Object cad)
    {
        System.out.println(cad.toString());
    }
:}


terminal String tint, tdouble, tchar, tbool, tstring, puntocoma, llav
ei, llaved, pari, pard, id, corchetei, corcheted, igual, print, print
ln, coma;
terminal String igualigual, desigual, mayor, menor, menorigual, mayor
igual, potencia , tvoid, nulo, tcomodin;
terminal String abstracto, caso, cat, clase, defecto, hacer, extiende
, final_, graph,msn;
terminal String importar, instanceof_, nuevo, privado, protegido, pub
lico, retorno, tcopi;

/*Palabras reservadas :v*/
terminal String tdefinir, tfusion, tconcatenar, tatexto, taentero, ta
decimal, tpeso, treservar, teql, twrite, twriteend, tclose;
terminal String repetir;
/*Casteos*/
terminal String leerarchivo,estatico, super_, switch_, este, tochar,
todouble, toint, str_, try_, escribir;
terminal String mientras, para, romper, continuar ,interrogante, dosp
untos, aumento, decremento, punto;
terminal String not, and, or, si, sinosi, sino, tostring, tolower, to
uper;
terminal String suma, menos, multi, div, xor, printable, modulo;
terminal String cadena;
terminal char caracter;
terminal int entero;
terminal double decimal;
terminal boolean booleano;

non terminal Bloque BLOQUE;
non terminal ArrayList<Nodo> LINST;
non terminal Nodo INST;
non terminal Declaracion DECLARACION;
```

```
non terminal Expresion EXP, AUMENTO, DECREMENTO, EXPLICITCAST;
non terminal INICIO;
non terminal Tipo PRIMITIVO;
non terminal Imprimir PRINT;
non terminal Message MSN;
non terminal ImprimirT PRINTABLE;
non terminal Asignacion ASIG;
non terminal While WHILE;
non terminal DoWhile DOWHILE;
non terminal Break BREAK;
non terminal Continuar CONTINUE;
non terminal If IF, ELSE;
non terminal Ternario TER;
non terminal For FOR;
non terminal Expresion /*ACTUALIZACION,*/ EXPRETORNO;
non terminal Nodo ACTUALIZACION;
/*---------------> Dimensiones*/
non terminal ArrayList<Expresion> NDIM, NDIMVACIO, NDIMVALOR;
/*--------------->*/
non terminal ArrayList<Expresion> LEXP, LVALORES;
non terminal AsignacionVector ASIGV;
non terminal NodoNario ITEM,LITEM,ARR;
non terminal Tipo TIPO;
non terminal ArrayList<Dec> LDEC;
non terminal Dec DEC, DECCONSTANTE;
non terminal ForEach FOREACH;
non terminal Caso CASO, DEFECTO;
non terminal ArrayList<Caso> LCASO;
non terminal Switch SWITCH;
non terminal ArrayList<String> LMODFICADOR;
non terminal String MODFUNCION, MODCONST;
non terminal Tipo RESULTADO;
non terminal ParametroFormal PARFORMAL;
non terminal ArrayList<ParametroFormal> LPARFORMAL;
/*non terminal Constructor CONSTRUCTOR;*/
non terminal Funcion FUNCION, CABECERAMET;
non terminal DeclaracionConstante DECLARACIONCONSTANTE;
/*non terminal ArrayList<Nodo> LFUNCION;*/
non terminal Retorno RETORNO;
non terminal Llamada LLAMADA;
non terminal String MODATRIB;
non terminal ArrayList<Expresion> PARACTUALES,ACTUALES;
non terminal Concatenar CONCATENAR;
```

```
/*Atributos de clase*/
/*non terminal ArrayList<String> LMODATRIB;*/
/*non terminal ArrayList<Nodo> LDECATRIB;*/
non terminal DeclaracionAtributo DECATRIB;

non terminal Fusion FUSION, CUERPOFUSION, CUERPO ;

/*Arreglos*/
non terminal Expresion VALORARREGLO ;
non terminal Copi COPI;
non terminal Expresion ACCESO, ORIGEN;
non terminal Expresion INSTANCIA;
non terminal Expresion VALORASIGNACION;
non terminal Importar IMPORTAR;


/*precedence left interrogante, punto;*/
precedence left or;
precedence left and;
precedence left xor;
precedence left igualigual, desigual;
precedence left menor, mayor, mayorigual, menorigual;
precedence left suma, menos;
precedence left multi, div, modulo;
precedence left potencia;
precedence left aumento, decremento ;
precedence left not;
precedence left interrogante;
precedence nonassoc pard , pari;
precedence nonassoc corcheted , corchetei;
/*
precedence left not;
precedence nonassoc aumento, decremento, not;
precedence right interrogante, dospuntos;
precedence left or;
precedence left and;
precedence left xor;
precedence left igualigual, desigual;
precedence left menor, menorigual, mayor, mayorigual, instanceof_;
precedence left suma, menos;
precedence left div, multi, modulo;
precedence left potencia;
```

```
precedence right nuevo;
precedence nonassoc pard , pari;
precedence nonassoc corcheted , corchetei;
precedence left punto;
*/
start with INICIO;

INICIO ::= LINST:lista {: Imprimir("Correcto"); raiz = new AST(lista)
;; :}
        ;

LINST::= LINST:lista INST:inst {: lista.add(inst); RESULT = lista;:}
        |INST:inst {: ArrayList<Nodo> l = new ArrayList<Nodo>(); l.ad
d(inst); RESULT = l;:}
;

INST::= PRINT:inst puntocoma {: RESULT = inst;:}
       |MSN:inst puntocoma {: RESULT = inst;:}
       |DECLARACION:inst puntocoma {: RESULT = inst;:}
       |ASIGV:inst puntocoma {: RESULT = inst;:}
       |ASIG:inst puntocoma {: RESULT = inst;:}
       |FUSION:inst puntocoma {: RESULT = inst;:}
       |COPI:inst puntocoma {: RESULT = inst;:}
       /*Todo esto debería de ser solo dentro de las funciones*/
       |FUNCION:inst  {: RESULT = inst;:}
       |IF:inst  {: RESULT = inst;:}
       |RETORNO:inst {:RESULT = inst;:}
       |LLAMADA:inst puntocoma {:RESULT = inst;:}
       |WHILE:inst  {:RESULT = inst;:}
       |DOWHILE:inst puntocoma {:RESULT = inst;:}
       |FOR:inst  {:RESULT = inst;:}
       |BREAK:inst puntocoma {:RESULT = inst;:}
       |CONTINUE:inst puntocoma {:RESULT = inst;:}
       |SWITCH:inst {:RESULT = inst;:}
       |DECLARACIONCONSTANTE:inst {:RESULT = inst;:}
       |AUMENTO:inst puntocoma{:RESULT =inst;:}
       |DECREMENTO:inst puntocoma{:RESULT =inst;:}
       |CONCATENAR:inst puntocoma{:RESULT =inst;:}
       |error
;

COPI::= tcopi:t pari EXP:origen coma EXP:destino pard
        {:
```

```
        RESULT = new Copi(origen, destino, tright, tleft);
        :}
;
                /*#definir id 10 */
                /*#definir id {{1,2},{2,3}} */
DECLARACIONCONSTANTE::= tdefinir:t DECCONSTANTE:nombre
        {:
            ArrayList<Dec> lista = new ArrayList<Dec>();
            lista.add(nombre);
            RESULT = new DeclaracionConstante(null, lista, tright, tl
eft);
        :}
;

DECCONSTANTE::=
      id:id VALORARREGLO:valor {: RESULT = new Dec(id, valor, idright
, idleft);:}
    | id:id VALORASIGNACION:valor
        {:
            if(valor instanceof Literal)
            {
                Literal valTmp = (Literal)valor;
                if(valTmp.tipo.isString())
                {
                    /*Hay que pasar la cadena a una expresionArreglo
*/
                    Arreglo tmpArreglo = new Arreglo();
                    tmpArreglo.columna = valorright;
                    tmpArreglo.linea = valorleft;
                    NodoNario raizArreglo = new NodoNario();
                    tmpArreglo.raiz = raizArreglo;
                    raizArreglo.hijos = new ArrayList<NodoNario>();
                    raizArreglo.tipo = new Tipo(Tipo.TypePrimitive.CH
AR);
                    char[] caracteres = valTmp.valor.toString().toCha
rArray();
                    for(char caracter : caracteres)
                    {
                        NodoNario nuevoNodo = new NodoNario();
                        nuevoNodo.tipo = tmpArreglo.tipo;
                        nuevoNodo.valor = caracter;
                        nuevoNodo.linea = valorright;
                        nuevoNodo.columna = valorleft;
```

```
                        raizArreglo.hijos.add(nuevoNodo);
                    }

                    RESULT = new Dec(id,
                            new ExpresionArreglo(raizArreglo, valorri
ght, valorleft)
                            ,idright,
                            idleft);
                }
                else
                {
                    RESULT = new Dec(id, valor, idright, idleft);
                }
            }
            else
            {
                RESULT = new Dec(id, valor, idright, idleft);
            }
        :}
;

/* ----------------------------------------> fin constantes -*/

DECLARACION::= TIPO:t LDEC:lista {: RESULT = new Declaracion(t, lista
, tright, tleft); :} ;

LDEC::= LDEC:lista  coma DEC:dec {: lista.add(dec); RESULT = lista;:}
        |DEC:dec {: ArrayList<Dec> lista = new ArrayList<Dec>(); list
a.add(dec); RESULT = lista;:}
;

DEC::=
      id:id {: RESULT = new Dec(id,idright, idleft);:}
    | id:id igual VALORASIGNACION:valor {: RESULT = new Dec(id,valor,
 idright, idleft);:}
    | id:id NDIM:dim {: RESULT = new Dec(id,dim, idright, idleft);:}
    | id:id NDIM:dim igual VALORARREGLO:valor {: RESULT = new Dec(id,
dim, valor, idright, idleft);:}
    | id:id NDIM:dim igual ARR:valor {: RESULT = new Dec(id,dim, new
ExpresionArreglo(valor, valorright, valorleft), idright, idleft);:}
    | id:id NDIM:dim igual EXP:valor
        {:
            if(valor instanceof Literal)
```

```
            {
                Literal valTmp = (Literal)valor;
                if(valTmp.tipo.isString())
                {
                    /*Hay que pasar la cadena a una expresionArreglo
*/
                    Arreglo tmpArreglo = new Arreglo();
                    tmpArreglo.columna = valorright;
                    tmpArreglo.linea = valorleft;
                    NodoNario raizArreglo = new NodoNario();
                    tmpArreglo.raiz = raizArreglo;
                    raizArreglo.hijos = new ArrayList<NodoNario>();
                    raizArreglo.tipo = new Tipo(Tipo.TypePrimitive.CH
AR);
                    char[] caracteres = valTmp.valor.toString().toCha
rArray();
                    for(char caracter : caracteres)
                    {
                        NodoNario nuevoNodo = new NodoNario();
                        nuevoNodo.tipo = tmpArreglo.tipo;
                        nuevoNodo.valor = caracter;
                        nuevoNodo.linea = valorright;
                        nuevoNodo.columna = valorleft;
                        raizArreglo.hijos.add(nuevoNodo);
                    }

                    RESULT = new Dec(id,dim,
                            new ExpresionArreglo(raizArreglo, valorri
ght, valorleft)
                            ,idright,
                            idleft);
                }
                else
                {
                    RESULT = new Dec(id,dim, valor, idright, idleft);
                }
            }
            else
            {
                RESULT = new Dec(id,dim, valor, idright, idleft);
            }
        :}
;
```

```
/*Dimensiones*/
NDIM ::=  NDIMVACIO:v {:RESULT =v; :}
        | LEXP:v {:RESULT =v; :}
;



NDIMVACIO::=
          NDIMVACIO:n corchetei corcheted {:RESULT = n ; n.add(null);
:}
        |corchetei corcheted {: RESULT = new ArrayList<Expresion>();
RESULT.add(null);:}
;

/*
NDIMVALOR::=
         NDIMVALOR:n corchetei EXP:valor corcheted {:RESULT = n ; n.a
dd(valor);:}
        |corchetei EXP:valor corcheted {: RESULT = new ArrayList<Expr
esion>(); RESULT.add(valor);:}
;
*/

VALORARREGLO::=
          llavei LEXP:l  llaved {: RESULT = new ExpresionArreglo(l,lr
ight,lleft);:}
        /*|nuevo:n id:tipo LEXP:l {: RESULT = new ExpresionArreglo(ne
w Tipo(tipo), l,nright,nleft);:}*/
;


ARR::= llavei LITEM:nodo llaved {: RESULT = nodo;:}
;

LITEM::= LITEM:nodo coma ITEM:item {: nodo.addHijo(item); RESULT = no
do; :}
        |ITEM:item  {: NodoNario nodo = new NodoNario(); nodo.addHijo
(item); RESULT = nodo;:}
;

ITEM::= EXP:exp {: RESULT = new NodoNario(exp);:}
        |ARR:exp {: RESULT = exp;:}
```

```
;

ASIG::= ACCESO:origen igual VALORASIGNACION:valor
        {:
            RESULT = new Asignacion(origen , valor, origenright, orig
enleft);
        :}

;

ASIGV::= id:id LEXP:coordenas igual EXP:valor {: RESULT = new Asignac
ionVector(id, coordenas, valor,  idright,idleft);:}
;


VALORASIGNACION::=
            EXP:exp {:RESULT = exp;:}
          |  INSTANCIA:exp  {:RESULT = exp;:}
;


INSTANCIA::=
        /*nuevo:n TIPO:tipo pari PARACTUALES:lista pard {: RESULT = n
ew Instancia(tipo, lista, nright, nleft);:}*/
        treservar:n pari EXP:valor pard {: RESULT = new Instancia(val
or, nright, nleft); :}
;

/*Lista de parametros actuales para llamadas a funciones*/
PARACTUALES::= ACTUALES:l{:RESULT = l;:}
            | {:RESULT = new ArrayList<Expresion>();:}
;
ACTUALES::=    ACTUALES:l coma EXP:exp {:l.add(exp);RESULT = l;:}
              |EXP:exp {:ArrayList<Expresion> l = new ArrayList<Expre
sion>(); l.add(exp); RESULT = l ;:}
;



PRINT::= print:p pari cadena:v pard
        {:
            RESULT = new Imprimir(
```

```
                    new Literal(new Tipo(Tipo.TypePrimitive.STRING),
v, vright,vleft),
                    pright,
                    pleft);
        :}
        |print:p pari cadena:v coma LVALORES:l pard
        {:
            RESULT = new Imprimir(
                    new Literal(new Tipo(Tipo.TypePrimitive.STRING),
v, vright,vleft),
                    pright,
                    pleft,
                    l);
        :}
;


MSN::= msn:p pari cadena:v pard
        {:
            RESULT = new Message(
                    new Literal(new Tipo(Tipo.TypePrimitive.STRING),
v, vright,vleft),
                    pright,
                    pleft);
        :}
        |msn:p pari cadena:v coma LVALORES:l pard
        {:
            RESULT = new Message(
                    new Literal(new Tipo(Tipo.TypePrimitive.STRING),
v, vright,vleft),
                    pright,
                    pleft,
                    l);
        :}
;




LVALORES ::= LVALORES:l coma EXP:exp {:RESULT = l; RESULT.add(exp); :
}
            |EXP:exp {: RESULT = new ArrayList<Expresion>(); RESULT.a
dd(exp); :}
```

```
;

LEXP::=   LEXP:l corchetei EXP:exp corcheted{:l.add(exp); RESULT = l;
:}
        | corchetei EXP:exp corcheted {: ArrayList<Expresion> l = new
 ArrayList<Expresion>(); l.add(exp); RESULT = l;:}
;

EXP::=
        /*Lógicas*/
         EXP:opi and EXP:opd {: RESULT = new And(opi, opd, opiright,o
pileft);:}
        |EXP:opi or EXP:opd {: RESULT = new Or(opi, opd, opiright,opi
left);:}
        |EXP:opi xor EXP:opd {: RESULT = new Xor(opi,opd,opiright,opi
left);:}
        /*Instanceof*/
        |EXP:op instanceof_ id:tipo {:RESULT = new InstanceOf(op,tipo
, opright, opleft);:}
        |not EXP:op {: RESULT = new Not(op, opright,opleft);:}

        /*Relacionales*/
        |EXP:opi igualigual EXP:opd {:RESULT = new Igual(opi,opd,opir
ight,opileft);:}
        |EXP:opi desigual EXP:opd {:RESULT = new Desigual(opi,opd,opi
right,opileft);:}
        |EXP:opi mayor EXP:opd {:RESULT = new Mayor(opi,opd,opiright,
opileft);:}
        |EXP:opi mayorigual EXP:opd {:RESULT = new MayorIgual(opi,opd
,opiright,opileft);:}
        |EXP:opi menor EXP:opd {:RESULT = new Menor(opi,opd,opiright,
opileft);:}
        |EXP:opi menorigual EXP:opd {:RESULT = new MenorIgual(opi,opd
,opiright,opileft);:}
        /*Ariteticas*/
        |EXP:opi suma EXP:opd {: RESULT = new Suma(opi,opd,opiright,o
pileft);:}
        |EXP:opi menos EXP:opd {: RESULT = new Resta(opi,opd,opiright
,opileft);:}
        |EXP:opi multi EXP:opd {: RESULT = new Multiplicacion(opi,opd
,opiright,opileft);:}
        |EXP:opi div EXP:opd {: RESULT = new Division(opi,opd,opirigh
t,opileft);:}
```

```
        |EXP:opi modulo EXP:opd {: RESULT = new Modulo(opi,opd,opirig
ht,opileft);:}
        |EXP:opi potencia EXP:opd {:RESULT = new Potencia(opi,opd,opi
right,opileft);:}
        |menos EXP:op {: RESULT = new Menos(op, opright,opleft);:}
        /*Ternario*/
        /*Aumento ++ y --*/
        |AUMENTO:op {:RESULT = op;:}
        |DECREMENTO:op {:RESULT = op;:}
        |LLAMADA:exp {:RESULT = exp;:}
        /*Primitivas*/
        |entero:v {: RESULT = new Literal(new Tipo(Tipo.TypePrimitive
.INT), v, vright,vleft);:}
        |decimal:v {: RESULT = new Literal(new Tipo(Tipo.TypePrimitiv
e.DOUBLE), v, vright,vleft);:}
        |booleano:v {: RESULT = new Literal(new Tipo(Tipo.TypePrimiti
ve.BOOL), v, vright,vleft);:}
        |cadena:v {: RESULT = new Literal(new Tipo(Tipo.TypePrimitive
.STRING), v, vright,vleft);:}
        |caracter:v {: RESULT = new Literal(new Tipo(Tipo.TypePrimiti
ve.CHAR), v, vright,vleft);:}
        /*ACCESO*/
        |ACCESO:v {:RESULT = v;:}
        |multi ACCESO:v {:RESULT = v;:}
        |pari EXP:op pard {: RESULT = op;:}
        |tatexto:t pari EXP:expresion pard {: RESULT = new aTexto(exp
resion, tright, tleft);:}
        /*|EXPLICITCAST:exp {:RESULT = exp;:}*/
        | nulo:r {:RESULT = new Literal(new Tipo(Tipo.TypePrimitive.N
ULO), null, rright, rleft);:}
        | tpeso:t pari id:nombre pard {: RESULT = new Peso(nombre,tri
ght, tleft );:}
        | CONCATENAR:v {:RESULT = v;:}
        | taentero:t pari EXP:expresion pard {: RESULT = new aEntero(
expresion, tright,tleft);:}
;

CONCATENAR::= tconcatenar:t pari EXP:exp1 coma EXP:exp2 pard {:RESULT
 = new Concatenar(exp1,exp2,tright,tleft);:}
;

AUMENTO::= ACCESO:op aumento {:RESULT = new Aumento(op, opright, ople
ft);:}
```

```
          /*|aumento EXP:op {:RESULT = new Preaumento(op, opright, op
left);:}       */
            ;
DECREMENTO::= ACCESO:op decremento {:RESULT = new Decremento(op, opri
ght, opleft);:}
              /*|decremento EXP:op {:RESULT = new Predecremento(op, o
pright, opleft);:}*/
;

ACCESO::= ACCESO:origen punto ORIGEN:destino {: RESULT = new Acceso(o
rigen, destino, origenright, origenleft);:}
        | ORIGEN:exp {:RESULT = exp;:}
;

ORIGEN::=
          TIPO:id {:RESULT = new Variable(id.nombreTipo(), idright, i
dleft);:}
        | ORIGEN:origen LEXP:coor {: RESULT = new AccesoVector(origen
 , coor , origenright, origenleft);:}

;

TIPO::= PRIMITIVO:t{:RESULT = t;:}
      | id:id {:RESULT = new Tipo(id,idright, idleft);:}
;

PRIMITIVO ::=
         tint:t {:RESULT = new Tipo(Tipo.TypePrimitive.INT,tright, tl
eft);:}
        |tchar:t {:RESULT =  new Tipo(Tipo.TypePrimitive.CHAR,tright,
 tleft);:}
        |tdouble:t {:RESULT = new Tipo(Tipo.TypePrimitive.DOUBLE,trig
ht, tleft);:}
        |tbool:t {:RESULT = new Tipo(Tipo.TypePrimitive.BOOL,tright,
tleft);:}
       /*|tstring:t {:RESULT = new Tipo(Tipo.TypePrimitive.STRING,tr
ight, tleft);:}*/
;



FUSION::= tfusion id:nombre llavei CUERPO:clase llaved
            {:
```

```
                clase.setId(nombre);
                RESULT = clase;
            :}
        | tfusion id:nombre llavei llaved
            {:
                RESULT = new Fusion();
                RESULT.setId(nombre);
            :}
;

CUERPO::=
          CUERPO:clase DECATRIB:f puntocoma
            {:
                clase.addAtributo(f);
                RESULT = clase;
            :}
        | DECATRIB:atributos puntocoma
            {:
                Fusion c = new Fusion(atributosright,atributosleft);
                c.addAtributo(atributos);
                RESULT = c;
            :}
;

DECATRIB::= TIPO:tipo LDEC:declaraciones
            {:
                Declaracion d  = new Declaracion(tipo,declaraciones,t
iporight,tipoleft);
                ArrayList<String> l = new ArrayList<String>();
                l.add("public");
                RESULT = new DeclaracionAtributo(l, d.tipo, d, tipori
ght,tipoleft);
            :}
;


FUNCION::= CABECERAMET:funcion BLOQUE:bloque {: funcion.setInstruccio
nes(bloque); RESULT = funcion;:}
;

CABECERAMET::=
      TIPO:res id:nombre pari LPARFORMAL:lf pard
                {:
```

```
                    Funcion f = new Funcion( new ArrayList<String>(),
res,nombre, lf, resright,resleft );
                    RESULT = f;
                :}
    | TIPO:res NDIMVACIO id:nombre pari LPARFORMAL:lf pard
                {:
                    Funcion f = new Funcion( new ArrayList<String>(),
res,nombre, lf, resright,resleft );
                    RESULT = f;
                :}
    | tvoid:res id:nombre pari LPARFORMAL:lf pard
                {:
                    Funcion f = new Funcion( new ArrayList<String>(),
 new Tipo(res),nombre, lf, resright,resleft );
                    RESULT = f;
                :}
;

BLOQUE::= llavei LINST:l llaved {:RESULT = new Bloque(l,lright,lleft)
;:}
        | llavei:l  llaved {:RESULT = new Bloque(new ArrayList<Nodo>(
),lright,lleft);:}
;


RETORNO ::= retorno:r EXPRETORNO:exp puntocoma{:RESULT = new Retorno(
exp, rright, rleft);:}
          | retorno:r puntocoma {: RESULT = new Retorno( rright, rlef
t);:}
;

EXPRETORNO::=
            EXP:exp {:RESULT = exp;:};

LPARFORMAL::=
         LPARFORMAL:lista coma PARFORMAL:par {:lista.add(par); RESULT
 = lista;:}
        |PARFORMAL:par {:ArrayList<ParametroFormal> l = new ArrayList
<ParametroFormal>(); l.add(par); RESULT = l;:}
        |{: ArrayList<ParametroFormal> l = new ArrayList<ParametroFor
mal>();RESULT = l; :}
;
```

```
PARFORMAL::=
         TIPO:tipo id:nombre{: RESULT = new ParametroFormal(tipo, nombre, true,new ArrayList<Expresion>(), tiporight, tipoleft);:}
        | final_:f TIPO:tipo id:nombre {: RESULT = new ParametroFormal(tipo, nombre, true,new ArrayList<Expresion>(),fright, fleft);:}
        | TIPO:tipo id:nombre NDIM:n{: RESULT = new ParametroFormal(tipo, nombre, true,n, tiporight, tipoleft);:}
        | final_:f TIPO:tipo id:nombre NDIM:n {: RESULT = new ParametroFormal(tipo, nombre, true, n, fright, fleft);:}
;


IF::= si:si pari EXP:condicion pard BLOQUE:bloque {:RESULT = new If(condicion, bloque, siright, sileft);:}
    | si:si pari EXP:condicion pard BLOQUE:bloque sino ELSE:inst2 {: RESULT= new If(condicion,bloque,inst2,siright, sileft);:}
;

ELSE::= IF:inst  {:RESULT = inst;:}
      | BLOQUE:bloque {: RESULT = new If(new Literal(new Tipo(Tipo.TypePrimitive.BOOL), true, bloqueright,bloqueleft),bloque, bloqueright, bloqueleft); :}
;

LLAMADA::=
            ACCESO:origen punto id:nombre pari PARACTUALES:l pard {:RESULT = new Llamada(origen,nombre, l, origenright,origenleft);:}

           | id:nombre pari PARACTUALES:l pard {:RESULT = new Llamada(null, nombre, l, nombreright,nombreleft);:}
;

WHILE::= mientras:inst pari EXP:condicion pard BLOQUE:bloque {: RESULT = new While(condicion,bloque,instright, instleft);:}
;

CONTINUE::= continuar:inst puntocoma {: RESULT = new Continuar(instright, instleft);:}
;

BREAK::= romper:inst puntocoma {: RESULT = new Break(instright, instleft);:}
;
```

```
SWITCH::= switch_:i pari EXP:condicion pard llavei LCASO:lista llaved
        {:
            RESULT = new Switch(condicion, lista, iright, ileft);
        :}
        | switch_:i pari EXP:condicion pard llavei LCASO:lista DEFECT
O:defecto llaved
        {:
            lista.add(defecto);
            RESULT = new Switch(condicion, lista, iright, ileft);
        :}
;

LCASO::= LCASO:l CASO:caso {:l.add(caso); RESULT = l;:}
        |CASO:caso {:ArrayList<Caso> l = new ArrayList<Caso>(); l.add
(caso); RESULT = l;:}
;

CASO::= caso:i EXP:condicion dospuntos LINST:linst {:RESULT = new Cas
o(condicion,new Bloque(linst,linstright,linstleft),iright,ileft);:}
;

DEFECTO::= defecto:i dospuntos LINST:linst {:RESULT = new Caso(null,n
ew Bloque(linst,linstright,linstleft),iright,ileft);:}
;

DOWHILE::=  hacer:i BLOQUE:bloque mientras:inst pari EXP:condicion pa
rd puntocoma
        {:
            RESULT = new DoWhile(condicion, bloque, iright, ileft);
        :}
;

FOR::=
      para:para pari DECLARACION:dec puntocoma EXP:condicion puntocom
a ACTUALIZACION:act  pard BLOQUE:bloque{: RESULT = new For(dec,condic
ion,act,bloque, paraleft,pararight);:}
    | para:para pari ASIG:dec puntocoma EXP:condicion puntocoma ACTUA
LIZACION:act  pard BLOQUE:bloque{: RESULT = new For(dec,condicion,act
,bloque,paraleft,pararight);:}
;

ACTUALIZACION::= AUMENTO :inst {:RESULT =inst;:}
```

```
                |DECREMENTO :inst {:RESULT =inst;:}
                |ASIG:inst {:RESULT =inst;:}
                |ASIGV:inst {:RESULT =inst;:}
;
```

```
package Analisis.olc;
import java_cup.runtime.Symbol;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import AST.AST;
import AST.Instruccion.Instruccion;
import Utilidades.ErrorC;
import Analisis.olc.Proyecto.*;
parser code
{:
    public ArrayList<ErrorC> listaErrores = new ArrayList();
    public ArrayList<Instruccion> ast = new ArrayList<Instruccion>();
    public Proyecto proyecto = null;

    public elemento tipoElemento;
    /*@Override
    public void syntax_error(Symbol cur_token)
    {
        List<Integer> listaIdTokens = expected_token_ids();
        LinkedList<String> listaNombres = new LinkedList<String>();
        for (Integer expected : listaIdTokens)
        {
            listaNombres.add(symbl_name_from_id(expected));
        }
        Utilidades.Singlenton.registrarError(String.valueOf(cur_token
.value),
            String.valueOf(cur_token.value) + ". Se esperaba :"+lista
Nombres.toString(), ErrorC.TipoError.LEXICO, cur_token.right+1, cur_t
oken.left+1);
        /*listaErrores.add(
```

```
                    listaErrores.add(new ErrorC(ErrorC.TipoError.SINT
ACTICO,
                                String.valueOf(cur_token.value) + ".
Se esperaba :"+listaNombres.toString(),
                                cur_token.right+1,
                                cur_token.left+1));

    }*/


    public void report_error(String message, Object info)
    {
        int linea = 0;
        int columna = 0;
        java_cup.runtime.Symbol s = null;
        StringBuilder m = new StringBuilder("Error Sintactico");

        if (info instanceof java_cup.runtime.Symbol)
        {
            s = ((java_cup.runtime.Symbol) info);
            if (s.left >= 0)
            {
                columna = s.left+1;
                if (s.right>= 0)
                {
                    linea = s.right+ 1;
                }
            }
        }

        m.append(" Se esperaba: "+message);
        //System.err.println(m.toString());
        //System.out.println("Error");
        //System.out.println("Error linea:"+linea+", col:"+columna);
        LinkedList<String> toks = new LinkedList();

        if(!expected_token_ids().isEmpty())
        {
            Imprimir("No esta vacia "+ expected_token_ids().size());

            for(int w=0; w<expected_token_ids().size(); w++)
            {
                if(expected_token_ids().get(w) !=sym.error)
```

```
                {
                    int tok = (int)expected_token_ids().get(w);

                    toks.add( symbol_name_from_id(tok) );
                }
            }
        }

        Imprimir(expected_token_ids().size());

        for(int w=0; w<expected_token_ids().size(); w++)
        {
            if(expected_token_ids().get(w)!=sym.error)
            {
                int tok = (int)expected_token_ids().get(w);
                toks.add( symbol_name_from_id(tok) );
            }
        }
        String esperados = "";
        for(String id : toks)
        {
            if(!esperados.equals(""))
            {
                esperados += ", ";
            }
            esperados += id;

        }
        Utilidades.Singlenton.registrarError(s.value.toString(), "Se
esperaba .. " +esperados, ErrorC.TipoError.SINTACTICO, linea, columna
);
    }

    public String symbol_name_from_id(int id){
        return sym.terminalNames[id];
    }

    /*public void addError(Symbol s)
    {
        listaErrores.add(new ErrorC("Sintactico",s.rights.right+1,Con
vertirObjectToString(s.value)));
    }*/
    public void Imprimir(Object cad)
```

```
    {
        System.out.println(cad.toString());
    }

:}


terminal String tproyecto, tconfiguracion, truta, tnombre, tcorrer, t
archivo, tfecha, tcarpeta;
terminal String coma, dospuntos, llavei, llaved, cadena;


non terminal INICIO;
non terminal Proyecto PROYECTO;
non terminal Proyecto CONTENIDO;
non terminal Proyecto ITEM;
non terminal String NOMBRE;
non terminal String CORRER;
non terminal Carpeta CONFIGURACION;
non terminal Carpeta ELEMENTOS;
non terminal Carpeta ELEMENTO;
non terminal Archivo ARCHIVO;
non terminal Archivo PROPSARCHIVO;
non terminal String PROPARCHIVO;
non terminal String FECHA;
non terminal Carpeta CARPETA;
non terminal String RUTA;

start with INICIO;

INICIO ::= llavei PROYECTO:pro llaved {: Imprimir("Correcto OLC."); p
royecto = pro; :}
            ;


PROYECTO ::= tproyecto dospuntos llavei CONTENIDO:proyecto llaved{:RE
SULT = proyecto;:}
            ;

CONTENIDO::= CONTENIDO:pro ITEM:proyTmp
            {:
                RESULT = pro;
                switch(tipoElemento)
```

```
                {
                    case RUTA:
                        pro.ruta = proyTmp.ruta;
                        break;
                    case NOMBRE:
                        pro.nombre = proyTmp.nombre;
                        break;
                    case CORRER:
                        pro.rutaRun = proyTmp.rutaRun;
                        break;
                    case CONF:
                        pro.carpeta = proyTmp.carpeta;
                        break;
                }
            :}
            | ITEM:pro {: RESULT= pro;:}
            ;

ITEM ::= RUTA:v {:RESULT = new Proyecto(); RESULT.ruta = v; tipoEleme
nto= Proyecto.elemento.RUTA;:}
        |NOMBRE:v {:RESULT = new Proyecto(); RESULT.nombre = v; tipoE
lemento = Proyecto.elemento.NOMBRE;:}
        |CORRER:v     {:RESULT = new Proyecto(); RESULT.rutaRun = v;
tipoElemento = Proyecto.elemento.CORRER;:}
        |CONFIGURACION:carpeta {:RESULT = new Proyecto(); RESULT.carp
eta = carpeta; tipoElemento = Proyecto.elemento.CONF;:}
        ;

RUTA ::= truta dospuntos cadena:cad coma {: RESULT = cad; tipoElement
o = Proyecto.elemento.RUTA;:}
;

NOMBRE ::= tnombre dospuntos cadena:cad coma
        {:
            RESULT = cad;
            tipoElemento = Proyecto.elemento.NOMBRE;
            Imprimir("Nombre elemento \t"+cad);
        :}
;

CORRER ::= tcorrer dospuntos cadena:cad coma {: RESULT = cad; tipoEle
mento = Proyecto.elemento.CORRER;:}
;
```

```
CONFIGURACION ::= tconfiguracion dospuntos llavei ELEMENTOS:carpeta l
laved
        {:
            RESULT = carpeta;
        :}
;

ELEMENTOS::= ELEMENTOS:carpeta coma ELEMENTO:carpeta2
            {:
                RESULT = carpeta;
                for(Carpeta car : carpeta2.subcarpetas)
                {
                    RESULT.subcarpetas.add(car);
                }
                for(Archivo arch : carpeta2.archivos)
                {
                    RESULT.archivos.add(arch);
                }
            :}
            | ELEMENTO:carpeta {: RESULT = carpeta;:}
;

ELEMENTO ::= ARCHIVO:archivo {:RESULT = new Carpeta(); RESULT.archivo
s.add(archivo);:}
            |CARPETA:carpeta {:RESULT = new Carpeta(); RESULT.subcarp
etas.add(carpeta);:}
;

ARCHIVO::=  tarchivo dospuntos llavei PROPSARCHIVO:archivo llaved {:
RESULT = archivo;:}
;

PROPSARCHIVO ::= PROPSARCHIVO:archivo PROPARCHIVO:v
                {:
                    RESULT = archivo;
                    if(tipoElemento == Proyecto.elemento.NOMBRE)
                    {
                        RESULT.nombre = v;
                    }
                    if(tipoElemento == Proyecto.elemento.FECHA_MOD)
                    {
                        RESULT.fecha =v;
```

```
                    }
                :}
                |PROPARCHIVO:v
                {:
                    RESULT = new Archivo();
                    if(tipoElemento == Proyecto.elemento.NOMBRE)
                    {
                        RESULT.nombre = v;
                    }
                    if(tipoElemento == Proyecto.elemento.FECHA_MOD)
                    {
                        RESULT.fecha =v;
                    }
                :}
;

PROPARCHIVO ::=  NOMBRE:v {: RESULT = v ; tipoElemento =Proyecto.elem
ento.NOMBRE;  Imprimir("Nombre\t"+v);:}
                |FECHA:v {: RESULT = v ; tipoElemento =Proyecto.eleme
nto.FECHA_MOD; Imprimir("Fecha\t"+v);:}
;

FECHA ::= tfecha dospuntos cadena:cad {:RESULT = cad; :}
;

CARPETA ::= tcarpeta dospuntos llavei NOMBRE:n ELEMENTOS:carpeta llav
ed
        {:
            RESULT = carpeta;
            RESULT.nombre = n;
        :}
;
```