U N I V E R S I D A D

D E G U A D A L A J A R A

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍA

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

Traductores de Lenguajes II

Actividad No. 04

Integrantes del equipo:

Nombre: Sandoval Marquez Anthony Esteven

Código: 215660767

Carrera: Ingeniería en Computación

Sección: D06

Nombre: Chávez Rojo Joaquín André

Código: 215662492

Carrera: Ingeniería en Computación

Sección: D06

Nombre: Becerra Gonzalez Diego Ivan

Código: 215661224

Carrera: Ingeniería en Computación

Sección: D04

Índice

Introducción

Como parte final del proyecto el producto final debe de ser capaz de leer instrucciones de lenguaje ensamblador y ejecutarlas cómo si se tratase de un emulador de mnemónicos.
En esta etapa tenemos que implementar un sistema embebido el cual sea capaz de distinguir entre el lenguaje máquina y el lenguaje de programación que hemos llevado a lo largo del proyecto.
Para el lenguaje máquina será necesario registrar todos los registros disponibles y que pueden ser usados en un emulador de procesador así como las distintas operaciones tanto de 1 o 2 operandos.

Objetivo

- **General**

  El objetivo de este trabajo es generar un compilador o traductor utilizando lenguajes de alto y bajo nivel, tablas de transiciones y diagramas de transición. El desarrollo completo deberá de ser por etapas, cada una con un propósito específico.
  En la última etapa el compilador será capaz de leer instrucciones en lenguaje máquina.

- **Particular**

  El objetivo particular de esta etapa es la implementación de una funcionalidad la cual permita al compilador leer instrucciones del lenguaje máquina de la misma forma que lee y traduce las instrucciones del lenguaje de programación que se ha llevado a lo largo del curso.
  De la misma manera que con el lenguaje de programación, el compilador tendrá que crear las producciones necesarias para poder traducir el lenguaje máquina y qué se le de una salida al usuario, ya sea de una compilación correcta o alguna excepción.
  Por ultimo, el compilador deberá ser capaz de ejecutar las instrucciones en lenguaje ensamblador previamente analizadas y comprobadas.

Desarrollo

El primer paso para la actualización del código fue agregar la regla semántica en dónde se comprueba la existencia de la librería ASM necesaria para poder traducir los mnemónicos operaciones que se pueden resolver con el compilador.
En este punto se hace la verificación de que exista la palabra reservada ASM y dependiendo del resultado podrá tomar dos caminos uno en el cual comenzará a leer todos los mnemónicos y por el contrario el camino del error será un error semántico que indique al usuario que nos ha importado a la librería correcta además de el número de línea dónde ocurre el error.

```
if(t.getLexicalComp().equals("ASM")){
    if(asmIndicator == true){
        mnems = new MNEMS(tokens.get(i+3).getLexeme(), tokens.get(i+4).getLexeme(),
tokens.get(i+6).getLexeme(), identifiers);
        mnems.checkMNEMS(regs, tokens, i, errors, p, struct);
    }else{
        errors.add(new ErrorLSSL(82, " --- Error Semantico({}): No se ha importado la
libreria ASM  [Linea: "+p.getLine()+", Caracter: "+p.getColumn()+"]", p, true));
    }
}
```

La segunda clase utilizada fue la clase para almacenar todos los registros disponibles y que el compilador puede leer y traducir.
En este punto se agregaron los registros básicos el acumulador, el registro base, el registro contador y el registro de datos. Estos registros a su vez pueden ser de 16 o de 8 bits por lo que se dividieron en un total de 12 registros correspondientes a 8 registros de 8 bits y 4 registros de 16 bits.

```
public class Registers {

    Hashtable<String, String> regs;

    public Registers(){
    }

    public void showRegisters(){
    }
}
```

El constructor de la clase de registros crea una nueva tabla hash e inserta todos los registros para que sea más fácil acceder a ellos mediante su nombre.

```
public Registers(){
        regs = new Hashtable<String, String>();

        regs.put("ax", "0");
        regs.put("bx", "0");
        regs.put("cx", "0");
        regs.put("dx", "0");

        regs.put("ah", "0");
        regs.put("al", "0");
```

```
        regs.put("bh", "0");
        regs.put("bl", "0");
        regs.put("ch", "0");
        regs.put("cl", "0");
        regs.put("dh", "0");
        regs.put("dl", "0");
    }
```

La función para mostrar registros devuelve los registros o almacenados en la tabla Hash con su respectivo índice de búsqueda.

```
public void showRegisters(){
        System.out.println("*** XX ***");
        System.out.println(" - ax: " + regs.get("ax"));
        System.out.println(" - bx: " + regs.get("bx"));
        System.out.println(" - cx: " + regs.get("cx"));
        System.out.println(" - dx: " + regs.get("dx"));

        System.out.println("*** XH XL ***");
        System.out.println(" - ah: " + regs.get("ah"));
        System.out.println(" - al: " + regs.get("al"));
        System.out.println(" - bh: " + regs.get("bh"));
        System.out.println(" - bl: " + regs.get("bl"));
        System.out.println(" - ch: " + regs.get("ch"));
        System.out.println(" - cl: " + regs.get("cl"));
        System.out.println(" - dh: " + regs.get("dh"));
        System.out.println(" - dl: " + regs.get("dl"));
    }
```

La clase MNEMS Almacena básicamente la estructura de todos los mnemónicos como por ejemplo la instrucción coma el valor 1 si es que se trata de una instrucción con sólo un operando y el valor 2 si se trata de una instrucción con más de un operando.
En su constructor básicamente se determina de qué mnemónico se trata para poder hacer la llamada a la función correspondiente y realizar la instrucción solicitada.
Entre las instrucciones que se agregaron fueron las instrucciones de MOV, ADD, SUB, MUL, DIV, POW, COS, SEN, TAN.
Ahora bien cada función contará con su propio proceso de comprobación sin embargo en todas se comprueba que los registros tengan el mismo tamaño es decir de 16 u 8 bits ambos.
Después de realizar estás comprobaciones se tendrá que establecer el límite que pueda almacenar cada registro. Es decir que si el registro tiene como máximo 8 bits podrá almacenar un valor de hasta 256.
De igual manera si se utilizan variables se tendrá que comprobar que el identificador existe por el contrario enviar al usuario un mensaje de error.

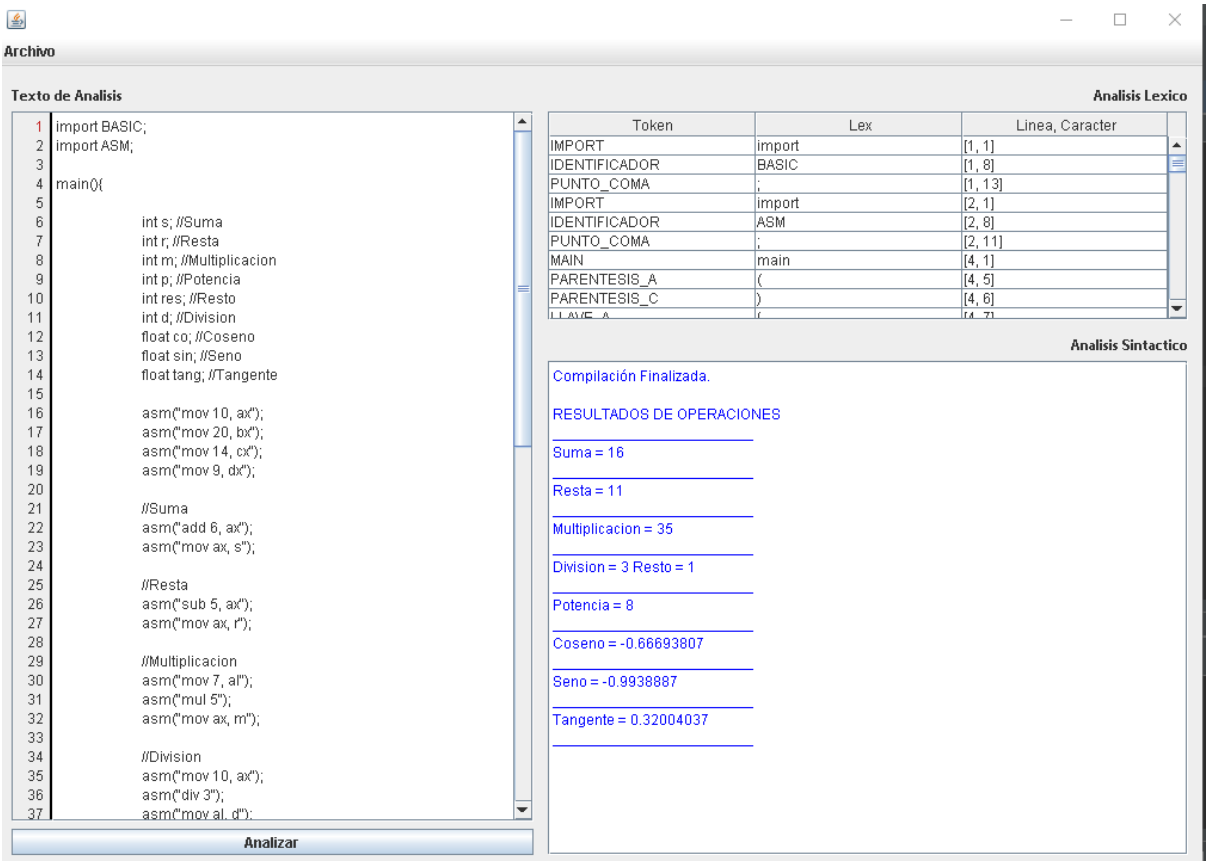
```
public class MNEMS {
    String mnem = "";
    String val1 = "";
    String val2 = "";

    String lexical1;
    String lexical2;
```

```
    ArrayList<Identifier> identifiers;

    public MNEMS(String mnem, String val1, String val2, ArrayList<Identifier>
identifiers)

    public void checkMNEMS(Registers regs, ArrayList<Token> t, int i,
ArrayList<ErrorLSSL> errors, Production p, String struct)


    public void MOV(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct)

    public void ADD(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct)

    public void SUB(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct)

    public void MUL(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct)

    public void DIV(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct)

    public void POW(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct)

    public void COS(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct)

    public void SEN(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct)

    public void TAN(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct)

    public boolean isReg(String value)

    public boolean isNum(String value)

    public boolean isIdent(String value)


    //Comprobar si ya existe un nombre de variable
    public boolean existIdentifier(String struct, String name)

    //Obtener una variable
    public Variable getVar(String struct, String name)

    //Busqueda de la estructura (main o alguna funcion) para comprobar si existe, es ese
caso devuelve el indice
    public int getIndexStruct(String struct)

}
```

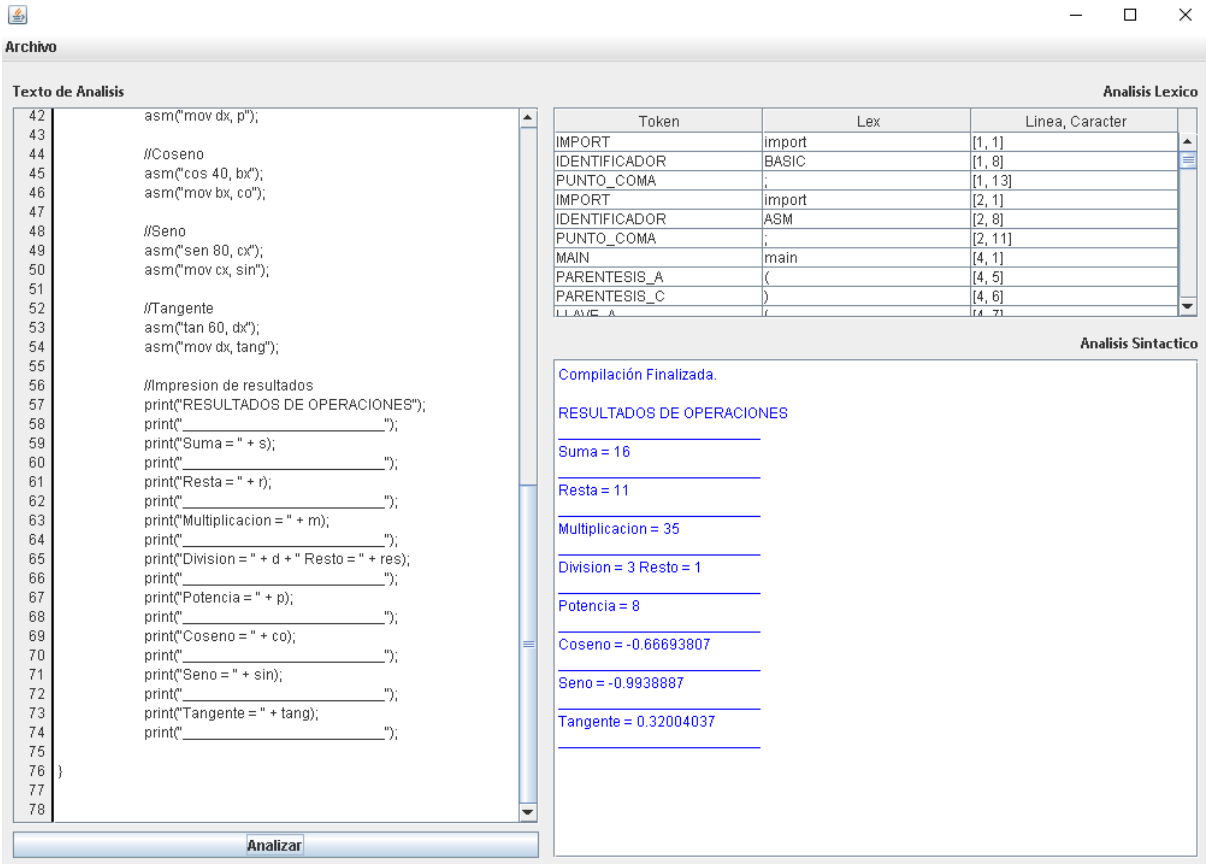
Por otra parte las funciones como “isReg”, “isNum” e “isIdent” realizarán comprobaciones para saber si se trata de un registro de 16 a 8 bits, si se trata de un número o en su caso de un identificador.

```
public boolean isReg(String value){
        if(value.equals("REG_16") || value.equals("REG_8")){
            return true;
        }
        return false;
    }
```

Por último se realizó un programa simulando algunas de las operaciones más importantes en una calculadora científica para mostrar el funcionamiento del compilador, todo esto claramente utilizando las instrucciones en ensamblador que agregamos.



En la parte izquierda se encuentra el código mezclado con ensamblador y en la derecha se encuentran los resultados de la compilación.



En esta otra imagen se observa lo restante del código en donde se colocaron las impresiones de los resultados.

Conclusiones

Cómo conclusión podemos agregar que la implementación de instrucciones en código máquina hace que el compilador sea más flexible emulando de cierta manera a compilador gcc qué es capaz de traducir y ejecutar instrucciones de lenguaje máquina incluso sí están de forma embebida con un programa en lenguaje C.

El resultado de este compilador trata de ser similar a los compiladores como los conocemos y se puede concluir que se obtuvieron nuevos conocimientos como por ejemplo el uso de expresiones regulares que ayudaron en gran medida a que el proyecto se desarrollará de forma más rápida además de recalcar la flexibilidad que tiene el lenguaje para poder utilizar librerías como por ejemplo aquellas que generan producciones para poder analizarlas de forma léxica y por lo tanto facilitar las etapas siguientes.

Al ser un proyecto por etapas facilita la creación de cada módulo además de que cada uno está bien delimitado por las funcionalidades que se deben de cumplir con cada entrega por lo que de cierta manera facilita la creación de un módulo individual que después será agregado a un proyecto más grande tal y como sucedería en la industria o en un ambiente laboral promedio.

El producto final se fue puliendo con cada paso de las distintas etapas por lo que además de ser un proyecto de calidad cumple con buenas prácticas de programación como por ejemplo comentarios en los distintos módulos del programa además de el testeo con las pruebas de caja negra y caja blanca.

Al final la documentación motiva a que el programa cumpla con ciertas características necesarias para poder ser un producto de calidad por lo tanto determinamos qué la creación de este tipo de documentos junto con los proyectos le dan un valor agregado además de necesario.

Bibliografías

¿Cuánto gana un programador? (s. f.). Guía Profesional de Indeed. Recuperado 12 de febrero de 2022, de https://mx.indeed.com/orientacion-profesional/pago-salario/cuanto-gana-programador

Como hacer un Compilador en Java | Analizador léxico, sintáctico y semántico. (2022, 20 enero). [Vídeo]. YouTube. https://www.youtube.com/watch?v=AHGe8l_yG6s&t=2963s

Apéndices

Clase “Registers.java”

```
package Code;
import java.util.Hashtable;
public class Registers {
    Hashtable<String, String> regs;

    public Registers(){
        regs = new Hashtable<String, String>();

        regs.put("ax", "0");
        regs.put("bx", "0");
        regs.put("cx", "0");
        regs.put("dx", "0");

        regs.put("ah", "0");
        regs.put("al", "0");
        regs.put("bh", "0");
        regs.put("bl", "0");
        regs.put("ch", "0");
        regs.put("cl", "0");
        regs.put("dh", "0");
        regs.put("dl", "0");
    }

    public void showRegisters(){
        System.out.println("*** XX ***");
        System.out.println(" - ax: " + regs.get("ax"));
        System.out.println(" - bx: " + regs.get("bx"));
        System.out.println(" - cx: " + regs.get("cx"));
        System.out.println(" - dx: " + regs.get("dx"));

        System.out.println("*** XH XL ***");
        System.out.println(" - ah: " + regs.get("ah"));
        System.out.println(" - al: " + regs.get("al"));
```

```
        System.out.println(" - bh: " + regs.get("bh"));
        System.out.println(" - bl: " + regs.get("bl"));
        System.out.println(" - ch: " + regs.get("ch"));
        System.out.println(" - cl: " + regs.get("cl"));
        System.out.println(" - dh: " + regs.get("dh"));
        System.out.println(" - dl: " + regs.get("dl"));
    }
}
```

Clase “MNEMS.java”

```
package Code;

import compilerTools.ErrorLSSL;
import compilerTools.Production;
import compilerTools.Token;
import java.util.ArrayList;

public class MNEMS {
    String mnem = "";
    String val1 = "";
    String val2 = "";

    String lexical1;
    String lexical2;

    ArrayList<Identifier> identifiers;

    public MNEMS(String mnem, String val1, String val2, ArrayList<Identifier>
identifiers){
        this.mnem = mnem;
        this.val1 = val1;
        this.val2 = val2;
        this.identifiers = identifiers;
    }

    public void checkMNEMS(Registers regs, ArrayList<Token> t, int i,
ArrayList<ErrorLSSL> errors, Production p, String struct){
        lexical1 = t.get(i+4).getLexicalComp();
        lexical2 = t.get(i+6).getLexicalComp();

        if(mnem.equals("mov")){
            MOV(regs, errors, p, t.get(i), struct);
        }else if(mnem.equals("add")){
            ADD(regs, errors, p, t.get(i), struct);
        }else if(mnem.equals("sub")){
            SUB(regs, errors, p, t.get(i), struct);
        }else if(mnem.equals("mul")){
            MUL(regs, errors, p, t.get(i), struct);
        }else if(mnem.equals("div")){
            DIV(regs, errors, p, t.get(i), struct);
        }else if(mnem.equals("pow")){
            POW(regs, errors, p, t.get(i), struct);
        }else if(mnem.equals("sqrt")){
            SQRT(regs, errors, p, t.get(i), struct);
```

```
        }else if(mnem.equals("cos")){
            COS(regs, errors, p, t.get(i), struct);
        }else if(mnem.equals("sen")){
            SEN(regs, errors, p, t.get(i), struct);
        }else if(mnem.equals("tan")){
            TAN(regs, errors, p, t.get(i), struct);
        }
    }


    public void MOV(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct){
        if(isReg(lexical1) && isReg(lexical2)){
            //revisar que sean del mismo tamaño
            if(lexical1.equals(lexical2))
                r.regs.put(val2, r.regs.get(val1));
            else
                errors.add(new ErrorLSSL(151, " --- Error Semantico({}): El tamaño de los
registros debe ser igual  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p,
true));
        }else if(isNum(lexical1) && isReg(lexical2)){
            //si es un registro de 8 bits no puede almacenar mas de 256
            if(lexical2.equals("REG_8") && Integer.parseInt(val1)>255)
                errors.add(new ErrorLSSL(152, " --- Error Semantico({}): El valor supera
la capacidad del registro  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p,
true));
            else
                r.regs.put(val2, val1);
        }else if(isIdent(lexical1) && isIdent(lexical2)){
            //comprobar que existen los identificadores
            if(existIdentifier(struct, val1) && existIdentifier(struct, val2)){
                getVar(struct, val2).saved = getVar(struct, val1).saved;
            }else{
                errors.add(new ErrorLSSL(153, " --- Error Semantico({}): Las variables no
existen  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }else if(isIdent(lexical1) && isReg(lexical2)){
            //Se obtiene valor de identificador, revisar si existe
            if(existIdentifier(struct, val1)){
                //si el registro es de 8 bits, revisar que el valor sea menor a 256
                if(lexical2.equals("REG_8") && Integer.parseInt(getVar(struct,
val1).saved)>255){
                    errors.add(new ErrorLSSL(154, " --- Error Semantico({}): El valor
supera la capacidad del registro  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]",
p, true));
                }else{
                    r.regs.put(val2, getVar(struct, val1).saved);
                }
            }else{
                errors.add(new ErrorLSSL(153, " --- Error Semantico({}): La variable no
existe  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }else if(isReg(lexical1) && isIdent(lexical2)){
            //Asignar el nuevo valor al identificador
            if(existIdentifier(struct, val2)){
                getVar(struct, val2).saved = r.regs.get(val1);
```

```
            }else{
                errors.add(new ErrorLSSL(150, " --- Error Semantico({}): La variable no
existe  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }else if(isNum(lexical1) && isIdent(lexical2)){
            //Revisar que el identificador exista
            if(existIdentifier(struct, val2)){
                getVar(struct, val2).saved = val1;
            }else{
                errors.add(new ErrorLSSL(155, " --- Error Semantico({}): La variable no
existe  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }else{
            //Error
            errors.add(new ErrorLSSL(160, " --- Error Semantico({}): La instruccion tiene
acciones no validas  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
        }
    }

    public void ADD(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct){
        int newValue = 0;
        //revisar que sean del mismo tamaño
        if(isReg(lexical1) && isReg(lexical2)){
            if(lexical1.equals(lexical2))
            {
                newValue = Integer.parseInt(r.regs.get(val2)) +
Integer.parseInt(r.regs.get(val1));
                r.regs.put(val2, String.valueOf(newValue));
            }
            else
            {
                errors.add(new ErrorLSSL(170, " --- Error Semantico({}): Los registros
deben ser del mismo tamaño  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p,
true));
            }

        }else if(isNum(lexical1) && isReg(lexical2)){
            if(lexical2.equals("REG_8") && Integer.parseInt(val1)>255){
                    errors.add(new ErrorLSSL(171, " --- Error Semantico({}): El valor
supera la capacidad del registro  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]",
p, true));
            }else{
                    newValue = Integer.parseInt(r.regs.get(val2)) +
Integer.parseInt(val1);
                    r.regs.put(val2, String.valueOf(newValue));
            }
        }else if(isIdent(lexical1) && isReg(lexical2)){
            //Se obtiene valor de identificador, revisar si existe
            if(existIdentifier(struct, val1)){
                //si el registro es de 8 bits, revisar que el valor sea menor a 256
                if(lexical2.equals("REG_8") && Integer.parseInt(getVar(struct,
val1).saved)>255){
                    errors.add(new ErrorLSSL(172, " --- Error Semantico({}): El valor
supera la capacidad del registro  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]",
p, true));
```

```
                }else{
                    newValue = Integer.parseInt(r.regs.get(val2)) +
Integer.parseInt(getVar(struct, val1).saved);
                    r.regs.put(val2, String.valueOf(newValue));
                }
            }else{
                errors.add(new ErrorLSSL(173, " --- Error Semantico({}): La variable no
existe  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }else if(isNum(lexical1) && isIdent(lexical2)){
            //Se obtiene valor de identificador, revisar si existe
            if(existIdentifier(struct, val2)){

                    newValue = Integer.parseInt(val1) + Integer.parseInt(getVar(struct,
val2).saved);
                    getVar(struct, val2).saved = Integer.toString(newValue);

            }else{
                errors.add(new ErrorLSSL(177, " --- Error Semantico({}): La variable no
existe  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }else if(isIdent(lexical1) && isIdent(lexical2)){
            //revisar si existen
            if(existIdentifier(struct, val1) && existIdentifier(struct, val2)){

                    newValue = Integer.parseInt(getVar(struct, val1).saved) +
Integer.parseInt(getVar(struct, val2).saved);
                    getVar(struct, val2).saved = Integer.toString(newValue);

            }else{
                errors.add(new ErrorLSSL(177, " --- Error Semantico({}): Alguna de las
variables no existe  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }else{
            //Error o caso no programado
            errors.add(new ErrorLSSL(176, " --- Error Semantico({}): La instruccion tiene
acciones no validas  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
        }
    }

    public void SUB(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct){
        int newValue = 0;
        //revisar que sean del mismo tamaño
        if(isReg(lexical1) && isReg(lexical2)){
            if(lexical1.equals(lexical2))
            {
                newValue = Integer.parseInt(r.regs.get(val2)) -
Integer.parseInt(r.regs.get(val1));
                r.regs.put(val2, String.valueOf(newValue));
            }
            else
            {
                errors.add(new ErrorLSSL(180, " --- Error Semantico({}): Los registros
deben ser del mismo tamaño  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p,
true));
```

```
            }

        }else if(isNum(lexical1) && isReg(lexical2)){
            newValue = Integer.parseInt(r.regs.get(val2)) - Integer.parseInt(val1);
            r.regs.put(val2, String.valueOf(newValue));

        }else if(isIdent(lexical1) && isReg(lexical2)){
            //Se obtiene valor de identificador, revisar si existe
            if(existIdentifier(struct, val1)){
                newValue = Integer.parseInt(r.regs.get(val2)) -
Integer.parseInt(getVar(struct, val1).saved);
                r.regs.put(val2, String.valueOf(newValue));

            }else{
                errors.add(new ErrorLSSL(182, " --- Error Semantico({}): La variable no
existe  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }else if(isNum(lexical1) && isIdent(lexical2)){
            //Se obtiene valor de identificador, revisar si existe
            if(existIdentifier(struct, val2)){

                    newValue = Integer.parseInt(getVar(struct, val2).saved) -
Integer.parseInt(val1) ;
                    getVar(struct, val2).saved = Integer.toString(newValue);

            }else{
                errors.add(new ErrorLSSL(183, " --- Error Semantico({}): La variable no
existe  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }else if(isIdent(lexical1) && isIdent(lexical2)){
            //revisar si existen
            if(existIdentifier(struct, val1) && existIdentifier(struct, val2)){

                    newValue = Integer.parseInt(getVar(struct, val1).saved) -
Integer.parseInt(getVar(struct, val2).saved);
                    getVar(struct, val2).saved = Integer.toString(newValue);

            }else{
                errors.add(new ErrorLSSL(184, " --- Error Semantico({}): Alguna de las
variables no existe  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }else{
            //Error o caso no programado
            errors.add(new ErrorLSSL(185, " --- Error Semantico({}): La instruccion tiene
acciones no validas  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
        }
    }

    public void MUL(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct){
        int newValue = 0;
        if(isReg(lexical1)){
            if(lexical1.equals("REG_8"))
            {
                newValue = Integer.parseInt(r.regs.get("al")) *
Integer.parseInt(r.regs.get(val1));
```

```
                r.regs.put("ax", String.valueOf(newValue));
            }
            else if(lexical1.equals("REG_16"))
            {
                newValue = Integer.parseInt(r.regs.get("ax")) *
Integer.parseInt(r.regs.get(val1));
                r.regs.put("dx", String.valueOf(newValue));
            }
        }else if(isNum(lexical1)){
            if(Integer.parseInt(val1) > 0)
            {
               if(Integer.parseInt(val1) < 256)
                {
                    newValue = Integer.parseInt(r.regs.get("al")) *
Integer.parseInt(val1);
                    r.regs.put("ax", String.valueOf(newValue));
                }
                else
                {
                    newValue = Integer.parseInt(r.regs.get("ax")) *
Integer.parseInt(val1);
                    r.regs.put("dx", String.valueOf(newValue));
                }
            }
            else{
                errors.add(new ErrorLSSL(190, " --- Error Semantico({}): No se aceptan
números negativos  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }
        else if(isIdent(lexical1)){
            if(existIdentifier(struct, val1)){
                int valorVariable = Integer.parseInt(getVar(struct, val1).saved);
                if(valorVariable > 0)
                {
                   if(valorVariable < 256)
                    {
                        newValue = Integer.parseInt(r.regs.get("al")) * valorVariable;
                        r.regs.put("ax", String.valueOf(newValue));
                    }
                    else
                    {
                        newValue = Integer.parseInt(r.regs.get("ax")) * valorVariable;
                        r.regs.put("dx", String.valueOf(newValue));
                    }
                }
                else{
                    errors.add(new ErrorLSSL(191, " --- Error Semantico({}): No se
aceptan números negativos  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p,
true));
                }
            }
            else{
                    errors.add(new ErrorLSSL(192, " --- Error Semantico({}): La variable
no existe  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }else{
```

```
        errors.add(new ErrorLSSL(193, " --- Error Semantico({}): La instruccion tiene
acciones no validas  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
        }
    }

    public void DIV(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct){
        int newValue = 0;
        int resto = 0;
        if(isReg(lexical1)){
            if(lexical1.equals("REG_8"))
            {
                newValue = Integer.parseInt(r.regs.get("ax")) /
Integer.parseInt(r.regs.get(val1));
                resto = Integer.parseInt(r.regs.get("ax")) %
Integer.parseInt(r.regs.get(val1));
                r.regs.put("al", String.valueOf(newValue));
                r.regs.put("ah", String.valueOf(resto));

                r.regs.put("ax", "0");
            }
            else if(lexical1.equals("REG_16"))
            {
                newValue = Integer.parseInt(r.regs.get("ax")) /
Integer.parseInt(r.regs.get(val1));
                resto = Integer.parseInt(r.regs.get("ax")) %
Integer.parseInt(r.regs.get(val1));
                r.regs.put("al", String.valueOf(newValue));
                r.regs.put("ah", String.valueOf(resto));

                r.regs.put("ax", "0");
            }
        }else if(isNum(lexical1)){
            if(Integer.parseInt(val1) > 0)
            {
               if(Integer.parseInt(val1) < 256)
                {
                    newValue = Integer.parseInt(r.regs.get("ax")) /
Integer.parseInt(val1);
                    resto = Integer.parseInt(r.regs.get("ax")) % Integer.parseInt(val1);
                    r.regs.put("al", String.valueOf(newValue));
                    r.regs.put("ah", String.valueOf(resto));

                    r.regs.put("ax", "0");
                }
                else
                {
                    newValue = Integer.parseInt(r.regs.get("ax")) /
Integer.parseInt(val1);
                    resto = Integer.parseInt(r.regs.get("ax")) % Integer.parseInt(val1);
                    r.regs.put("al", String.valueOf(newValue));
                    r.regs.put("ah", String.valueOf(resto));

                    r.regs.put("ax", "0");
                }
            }
```

```
            else{
                errors.add(new ErrorLSSL(190, " --- Error Semantico({}): No se aceptan
números negativos  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }
        else if(isIdent(lexical1)){
            if(existIdentifier(struct, val1)){
                int valorVariable = Integer.parseInt(getVar(struct, val1).saved);
                if(valorVariable > 0)
                {
                   if(valorVariable < 256)
                    {
                        newValue = Integer.parseInt(r.regs.get("ax")) / valorVariable;
                        resto = Integer.parseInt(r.regs.get("ax")) % valorVariable;
                        r.regs.put("al", String.valueOf(newValue));
                        r.regs.put("ah", String.valueOf(resto));

                        r.regs.put("ax", "0");
                    }
                    else
                    {
                        newValue = Integer.parseInt(r.regs.get("ax")) / valorVariable;
                        resto = Integer.parseInt(r.regs.get("ax")) % valorVariable;
                        r.regs.put("al", String.valueOf(newValue));
                        r.regs.put("ah", String.valueOf(resto));

                        r.regs.put("ax", "0");
                    }
                }
                else{
                    errors.add(new ErrorLSSL(191, " --- Error Semantico({}): No se
aceptan números negativos  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p,
true));
                }
            }
            else{
                    errors.add(new ErrorLSSL(192, " --- Error Semantico({}): La variable
no existe  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }else{
        errors.add(new ErrorLSSL(193, " --- Error Semantico({}): La instruccion tiene
acciones no validas  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
        }
    }

    public void POW(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct){
        int pot = 0;
        if(isNum(lexical1) && isNum(lexical2)){
            pot = (int) Math.pow(Integer.valueOf(val1), Integer.valueOf(val2));
            r.regs.put("dx", String.valueOf(pot));
        }else if(isReg(lexical1) && isNum(lexical2)){
            pot = (int) Math.pow(Integer.valueOf(r.regs.get(val1)),
Integer.valueOf(val2));
            r.regs.put("dx", String.valueOf(pot));
        }else{
```

```
            errors.add(new ErrorLSSL(220, " --- Error Semantico({}): La instruccion tiene
acciones no validas  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
        }
    }

   public void SQRT(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct){
        int sqrt = 0;
        if(isNum(lexical1)){
            sqrt = (int) Math.sqrt(Integer.valueOf(val1));
            r.regs.put("dx", String.valueOf(sqrt));
        }else if(isReg(lexical1)){
            sqrt = (int) Math.sqrt(Integer.valueOf(r.regs.get(val1)));
            r.regs.put("dx", String.valueOf(sqrt));
        }else if(isIdent(lexical1)){
            if(existIdentifier(struct, val1)){
                int valorVariable = Integer.parseInt(getVar(struct, val1).saved);
                if(valorVariable > 0)
                {
                    sqrt = (int) Math.sqrt(valorVariable);
                    r.regs.put("dx", String.valueOf(sqrt));
                }
                else{
                    errors.add(new ErrorLSSL(191, " --- Error Semantico({}): No se
aceptan números negativos  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p,
true));
                }
            }
            else{
                    errors.add(new ErrorLSSL(192, " --- Error Semantico({}): La variable
no existe  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }else{
            errors.add(new ErrorLSSL(230, " --- Error Semantico({}): La instruccion tiene
acciones no validas  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
        }
    }


    public void COS(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct){
        float cos = 0;
        double value = 0;
        if(isNum(lexical1) && isReg(lexical2)){
            value = Double.parseDouble(val1);
            cos =  (float)Math.cos(value);
            r.regs.put(val2, String.valueOf(cos));
        }else if(isReg(lexical1) && isReg(lexical2)){
            value = Double.parseDouble(r.regs.get(val1));
            cos =  (float)Math.cos(value);
            r.regs.put(val2, String.valueOf(cos));
        }else if(isReg(lexical1) && isIdent(lexical2)){
            value = Double.parseDouble(r.regs.get(val1));
            cos =  (float)Math.cos(value);
            //Se obtiene valor de identificador, revisar si existe
            if(existIdentifier(struct, val2)){
```

```
                getVar(struct, val2).saved = String.valueOf(cos);
            }else{
                errors.add(new ErrorLSSL(183, " --- Error Semantico({}): La variable no
existe  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }else{
            errors.add(new ErrorLSSL(221, " --- Error Semantico({}): La instruccion tiene
acciones no validas  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
        }
    }

    public void SEN(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct){
        float sen = 0;
        double value = 0;
        if(isNum(lexical1) && isReg(lexical2)){
            value = Double.parseDouble(val1);
            sen =  (float)Math.sin(value);
            r.regs.put(val2, String.valueOf(sen));
        }else if(isReg(lexical1) && isReg(lexical2)){
            value = Double.parseDouble(r.regs.get(val1));
            sen =  (float)Math.sin(value);
            r.regs.put(val2, String.valueOf(sen));
        }else if(isReg(lexical1) && isIdent(lexical2)){
            value = Double.parseDouble(r.regs.get(val1));
            sen =  (float)Math.sin(value);
            //Se obtiene valor de identificador, revisar si existe
            if(existIdentifier(struct, val2)){
                getVar(struct, val2).saved = String.valueOf(sen);
            }else{
                errors.add(new ErrorLSSL(184, " --- Error Semantico({}): La variable no
existe  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }else{
            errors.add(new ErrorLSSL(222, " --- Error Semantico({}): La instruccion tiene
acciones no validas  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
        }
    }

    public void TAN(Registers r, ArrayList<ErrorLSSL> errors, Production p, Token t,
String struct){
        float tan = 0;
        double value = 0;
        if(isNum(lexical1) && isReg(lexical2)){
            value = Double.parseDouble(val1);
            tan =  (float)Math.tan(value);
            r.regs.put(val2, String.valueOf(tan));
        }else if(isReg(lexical1) && isReg(lexical2)){
            value = Double.parseDouble(r.regs.get(val1));
            tan =  (float)Math.tan(value);
            r.regs.put(val2, String.valueOf(tan));
        }else if(isReg(lexical1) && isIdent(lexical2)){
            value = Double.parseDouble(r.regs.get(val1));
            tan =  (float)Math.tan(value);
            //Se obtiene valor de identificador, revisar si existe
            if(existIdentifier(struct, val2)){
```

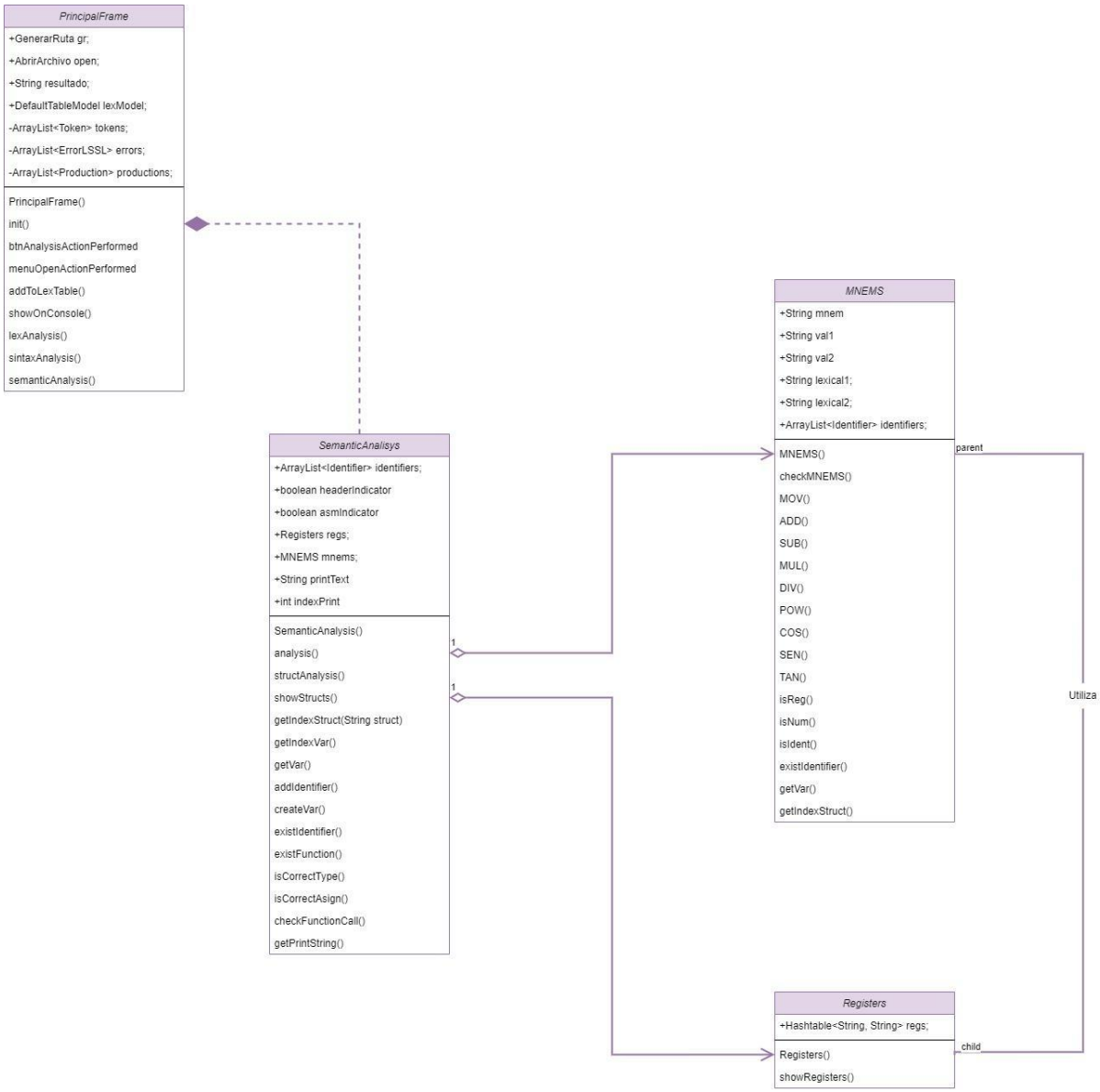
```
                getVar(struct, val2).saved = String.valueOf(tan);
            }else{
                errors.add(new ErrorLSSL(183, " --- Error Semantico({}): La variable no
existe  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
            }
        }else{
            errors.add(new ErrorLSSL(221, " --- Error Semantico({}): La instruccion tiene
acciones no validas  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
        }
    }

    public boolean isReg(String value){
        if(value.equals("REG_16") || value.equals("REG_8")){
            return true;
        }
        return false;
    }

    public boolean isNum(String value){
        if(value.equals("NUMERO")){
            return true;
        }
        return false;
    }

    public boolean isIdent(String value){
        if(value.equals("IDENTIFICADOR")){
            return true;
        }
        return false;
    }


    //Comprobar si ya existe un nombre de variable
    public boolean existIdentifier(String struct, String name){
        int index = getIndexStruct(struct);
        ArrayList<Variable> idfs = identifiers.get(index).words;
        for(int i = 0; i < idfs.size(); i++){
            if(name.equals(idfs.get(i).name)){
                return true;
            }
        }
        return false;
    }

    //Obtener una variable
    public Variable getVar(String struct, String name){
        int index = getIndexStruct(struct);
        ArrayList<Variable> idfs = identifiers.get(index).words;
        for(int i = 0; i < idfs.size(); i++){
            if(idfs.get(i).name.equals(name)){
                return idfs.get(i);
            }
        }
        return null;
    }
```

```
    //Busqueda de la estructura (main o alguna funcion) para comprobar si existe, es ese
caso devuelve el indice
    public int getIndexStruct(String struct){
        for(int i = 0; i < identifiers.size(); i++){
            if(identifiers.get(i).structName.equals(struct)){
                return i;
            }
        }
        return -1;
    }

}
```

Acrónimos

No se usaron acrónimos en esta versión del documento.

Diagrama(de clases)



Link de diagrama:
https://drive.google.com/file/d/14JpBNvA4SEJxv_sYgJeso6fHc0II6BFa/view?usp=sharing

Requisitos funcionales

RF1 - Deberá de permitir el ingreso de texto.
RF2 - Deberá de realizar el análisis léxico.
RF3 - Deberá de poder verificar si una instrucción es sintácticamente correcta o no.
RF4 - En caso de que haya un error sintáctico, debe de indicar cual es el error.
RF5 - Deberá de poder verificar si el texto ingresado es semánticamente correcto.
RF6 - Mantener el orden de las variables en los distintos ámbitos del programa (main, global, funcion)
RF7 - Mostrar la posición de errores semánticos e información correspondiente.
RF8 - El compilador debe ser capaz de leer y ejecutar instrucciones de lenguaje ensamblador

RF9 - El compilador debe validar léxica, sintáctica y semánticamente del código máquina o ensamblador.
RF10 - El compilador debe aceptar lenguaje embebido

Requisitos no funcionales

RNF1 - Disponibilidad
Mantener la disponibilidad del servicio al usuario después de la ejecución de un análisis completo.

RNF2 - Cantidad de palabras
Permitir al usuario el análisis de 1 o más tokens. El usuario podrá evaluar un token a la vez o será posible ingresar más de un token para su evaluación obteniendo el análisis de cada uno de ellos.

RNF3 - Manejo de memoria
El programa será capaz de liberar la memoria no utilizada al término del mismo además de utilizar estructuras de datos dinámicas para el almacenamiento de los datos en tiempo de ejecución.

RFN4 - Mensajes informativos.
El programa debera de mostrar el mensaje de analisis exitoso en color azul y en caso de que haya errores, el mensaje debera ser de color rojo.

RFN 5 - Indicador de errores.
En caso de que haya un error, se deberá de indicar la línea donde se encuentra el error, en el área donde se ingresa el código.

Complejidad Ciclomática

Para esta sección se utilizará la fórmula:
V(G) = Aristas – Nodos + 2

Complejidad ciclomática = 100 - 101 + 2 = 1.

Esto indica que es un código simple, esto gracias al uso de la librería JFLEX, que facilitó más el proceso del análisis léxico, a comparación de la práctica anterior.

COCOMO

Para este proyecto utilizamos el COCOMO básico en su modo Semi-Orgánico donde:

a	b	c	d
3.00	1.12	2.50	0.35

1.- Estimación de la cantidad de instrucciones

Líneas = L = 120 * entradas/salidas

L = 120 * 14 aprox
L = 1680
miles de líneas = ML = 1.68

2.- Calcular personas necesarias para llevar a cabo el proyecto
MM = a(ML^b)

MM = 3 * (1.68 ^1.12)
MM = 3 * 1.78
MM = 5.34 ~ 6 Personas/Mes

3.- Calcular el tiempo de desarrollo del proyecto
TDEV = c*(MM^d)

TDVE = 2.5 * (5.34^0.35)
TDVE = 2.5 * 1.79
TDVE = 4.49 ~ 5 meses

4.- Calcular personas necesarias para realizar el proyecto
CosteH = MM / TDVE

CosteH = 5.34 / 4.49
CosteH = 1.189 ~ 2 personas

5.- Calcular la productividad
P = L / MM
Productividad = 1680 / 5.34
Productividad = 314.6 instrucciones/persona_mes

6.- Calcular el costo total del proyecto
CosteM = costeH * salario promedio entre programadores

Salario promedio programador de software = $14,439 al mes

costeM = 1.189 * $14439

costeM = $17,167.9

Caja Negra

Se especificó la extensión de los archivos para diferenciarlos entre sí.

Número del caso de prueba	Componente	Descripción de lo que se probará	Prerrequisitos
V1	Variable.java	Generar el registro de las variables del texto	N/A
I1	Identifier.java	Guarda las variables de cada entorno así como los parámetros si se trata de una función	Variable.java
P1	PrincipalFrame	Muestra los tokens generados por el analisis lexico en la tabla TableLex	Lexer.flex, Lexer.java
P2	PrincipalFrame	Muestra el resultado de la compilación, errores y mensajes para el usuario en el panel TextSintax	SintaticAnalysis SemanticAnalysis
S1	SintaticAnalysis	Realiza el análisis sintáctico correctamente	Lexer.flex, Lexer.java
SE1	SemanticAnalysis	Realiza el análisis semántico correctamente	Variable.java Identifier.java
R1	Registers.java	Cargar los registros de 16 y 8 bits en la hashtable	N/A
M1	MNEMS.java	Crear la instancia de un mnemónico	N/A
M2	MNEMS.java	Realizar chequeo de mnemónico para asociarlo a una instrucción	Registers.java
M3	MNEMS.java	Realizar las comprobaciones del mnemónico y agregarlo a la hashtable	Registers.java

Número de caso de prueba	V1
Proceso	Generar el registro de las variables del texto
Variable	Tokens
Descripción	Se toma la producción para registrar las variables y tipo
Caso válido	Se registra la nueva variable
Caso no válido	Se descarta el registro de la variable
Observaciones	La creación se hace correctamente

Número de caso de prueba	I1
Proceso	Almacenar las variables
Variable	Variable
Descripción	Guarda las variables de cada entorno así como los parámetros si se trata de una función
Caso válido	Se almacenan las variables de cada contexto en una lista dinámica
Caso no válido	Las variables no son registradas, devolviendo error
Observaciones	La operación se cumple correctamente

Número de caso de prueba	P1
Proceso	Demostración de los tokens identificados
Variable	tokens
Descripción	Los tokens identificados y almacenados en el arreglo se muestran en la tabla
Caso válido	Los tokens son mostrados
Caso no válido	Los tokens no se muestran, o se muestran en un orden incorrecto

Observaciones	Los tokens se muestran de manera correcta

Número de caso de prueba	P2
Proceso	Demostración del resultado del análisis semántico
Variable	SemanticAnalysis
Descripción	Se muestra el resultado del analisis semantico por SemanticAnalysis.java
Caso válido	Se muestran los resultados de la compilación así como errores en caso de haber alguno (mensaje, fila y columna)
Caso no válido	No se indica nada, o se indica un caso contrario al que en realidad sucede
Observaciones	El análisis semántico se hace correctamente

Número de caso de prueba	SE1
Proceso	Ejecución del análisis semántico
Variable	Variable.java Identifier.java
Descripción	Los resultados de la ejecución son almacenados para ser mostrados al usuario
Caso válido	Los resultados son visibles
Caso no válido	No se muestra ningún resultado
Observaciones	Los resultados se muestran correctamente

Número de caso de prueba	R1
Proceso	Cargar los registros de 16 y 8 bits en la hashtable
Variable	N/A
Descripción	El programa carga como índices en la

	hashtable los registros de 16 y 8 bits (AX, BX, CX, DX, AH, AL, BH, BL, CH, CL, DH, DL)
Caso válido	Los registros son cargados a la hashtable
Caso no válido	Error en la creación de la hashtable
Observaciones	Los registros se cargan correctamente

Número de caso de prueba	M1
Proceso	Crear la instancia de un mnemónico
Variable	N/A
Descripción	El programa crea la estructura general de un mnemónico donde se incluye la instrucción, el operando 1 y en algunos casos el operando 2. Además de una lista de identificadores
Caso válido	Creación y comprobación del mnemónico correctamente
Caso no válido	Error en la instanciación
Observaciones	Los mnemónicos se cargan correctamente

Número de caso de prueba	M2
Proceso	Realizar chequeo de mnemónico para asociarlo a una instrucción
Variable	Registers.java
Descripción	El programa separa el mnemónico de acuerdo a la instrucción que contenga para después realizar la comprobación
Caso válido	Validación de la instrucción
Caso no válido	Instrucción no encontrada
Observaciones	N/A

Número de caso de prueba	M3
Proceso	Realizar las comprobaciones del mnemónico y agregarlo a la hashtable
Variable	Registers.java
Descripción	El programa realiza la comprobación semántica de la instrucción. (Comprobar el tamaño de los registros 16 u 8 bits, comprobar identificadores de ser necesario)
Caso válido	Registro de la instrucción
Caso no válido	Error en la estructura de la instrucción
Observaciones	Comprobaciones realizadas con éxito para cada instrucción

Caja Blanca

```
//Condicion 1
if(t.getLexicalComp().equals("ASM")){
    //Condicion 2
    if(asmIndicator == true){
       mnems = new MNEMS(tokens.get(i+3).getLexeme(),
tokens.get(i+4).getLexeme(), tokens.get(i+6).getLexeme(), identifiers);
       mnems.checkMNEMS(regs, tokens, i, errors, p, struct);
    }else{
       errors.add(new ErrorLSSL(82, " --- Error Semantico({}): No se ha
importado la libreria ASM  [Linea: "+p.getLine()+", Caracter:
"+p.getColumn()+"]", p, true));
    }
}
```

Condicion 1 - t.getLexicalComp().equals("ASM")	
Falso	N/A
Verdadero	Condición 2

Condicion 2 - asmIndicator == true	
Falso	Error semantico
Verdadero	Creacion de objeto MNEMS

```
//Condicion 1
if(mnem.equals("mov")){
    MOV(regs, errors, p, t.get(i), struct);
//Condicion 2
}else if(mnem.equals("add")){
    ADD(regs, errors, p, t.get(i), struct);
//Condicion 3
}else if(mnem.equals("sub")){
    SUB(regs, errors, p, t.get(i), struct);
//Condicion 4
}else if(mnem.equals("mul")){
    MUL(regs, errors, p, t.get(i), struct);
//Condicion 5
}else if(mnem.equals("div")){
    DIV(regs, errors, p, t.get(i), struct);
//Condicion 6
}else if(mnem.equals("pow")){
    POW(regs, errors, p, t.get(i), struct);
//Condicion 7
}else if(mnem.equals("cos")){
    COS(regs, errors, p, t.get(i), struct);
//Condicion 8
}else if(mnem.equals("sen")){
    SEN(regs, errors, p, t.get(i), struct);
//Condicion 9
}else if(mnem.equals("tan")){
    TAN(regs, errors, p, t.get(i), struct);
}
```

Condicion 1 - mnem.equals("mov")	
Falso	Condicion 2
Verdadero	Comprobar estructura de instrucción MOV

Condicion 2 - mnem.equals("add")

Falso	Condicion 3
Verdadero	Comprobar estructura de instrucción ADD

Condicion 3 - mnem.equals("sub")	
Falso	Condicion 4
Verdadero	Comprobar estructura de instrucción SUB

Condicion 4 - mnem.equals("mul")	
Falso	Condicion 5
Verdadero	Comprobar estructura de instrucción MUL

Condicion 5 - mnem.equals("div")	
Falso	Condicion 6
Verdadero	Comprobar estructura de instrucción DIV

Condicion 6 - mnem.equals("pow")	
Falso	Condicion 7
Verdadero	Comprobar estructura de instrucción POW

Condicion 7 - mnem.equals("sen")	
Falso	Condicion 8
Verdadero	Comprobar estructura de instrucción SEN

Condicion 8 - mnem.equals("cos")	
Falso	Condicion 9
Verdadero	Comprobar estructura de instrucción COS

Condicion 9 - mnem.equals("tan")	
Falso	N/A
Verdadero	Comprobar estructura de instrucción TAN

```
//Condicion 1
if(isReg(lexical1) && isReg(lexical2)){
    //revisar que sean del mismo tamaño
    //Condicion 7
    if(lexical1.equals(lexical2))
        r.regs.put(val2, r.regs.get(val1));
    else
        errors.add(new ErrorLSSL(151, " --- Error Semantico({}): El tamaño de los
registros debe ser igual  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p,
true));
//Condicion 2
}else if(isNum(lexical1) && isReg(lexical2)){
    //si es un registro de 8 bits no puede almacenar mas de 256
    //Condicion 8
    if(lexical2.equals("REG_8") && Integer.parseInt(val1)>255)
        errors.add(new ErrorLSSL(152, " --- Error Semantico({}): El valor supera la
capacidad del registro  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p,
true));
    else
        r.regs.put(val2, val1);
//Condicion 3
}else if(isIdent(lexical1) && isIdent(lexical2)){
    //comprobar que existen los identificadores
    //Condicion 9
    if(existIdentifier(struct, val1) && existIdentifier(struct, val2)){
        getVar(struct, val2).saved = getVar(struct, val1).saved;
    }else{
        errors.add(new ErrorLSSL(153, " --- Error Semantico({}): Las variables no existen
[Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
    }
//Condicion 4
}else if(isIdent(lexical1) && isReg(lexical2)){
    //Se obtiene valor de identificador, revisar si existe
    //Condicion 10
    if(existIdentifier(struct, val1)){
        //si el registro es de 8 bits, revisar que el valor sea menor a 256
```

```
        //Condicion 13
        if(lexical2.equals("REG_8") && Integer.parseInt(getVar(struct, val1).saved)>255){
            errors.add(new ErrorLSSL(154, " --- Error Semantico({}): El valor supera la
capacidad del registro  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p,
true));
        }else{
            r.regs.put(val2, getVar(struct, val1).saved);
        }
    }else{
        errors.add(new ErrorLSSL(153, " --- Error Semantico({}): La variable no existe
[Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
    }
//Condicion 5
}else if(isReg(lexical1) && isIdent(lexical2)){
    //Asignar el nuevo valor al identificador
    //Condicion 11
    if(existIdentifier(struct, val2)){
        getVar(struct, val2).saved = r.regs.get(val1);
    }else{
        errors.add(new ErrorLSSL(150, " --- Error Semantico({}): La variable no existe
[Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
    }
//Condicion 6
}else if(isNum(lexical1) && isIdent(lexical2)){
    //Revisar que el identificador exista
    //Condicion 12
    if(existIdentifier(struct, val2)){
        getVar(struct, val2).saved = val1;
    }else{
        errors.add(new ErrorLSSL(155, " --- Error Semantico({}): La variable no existe
[Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
    }
}else{
    //Error
    errors.add(new ErrorLSSL(160, " --- Error Semantico({}): La instruccion tiene
acciones no validas  [Linea: "+t.getLine()+", Caracter: "+t.getColumn()+"]", p, true));
}
```

Condicion 1 - `isReg(lexical1) && isReg(lexical2)`	
Falso	Condicion 2
Verdadero	Condicion 7

Condicion 2 - `isNum(lexical1) && isReg(lexical2)`	
Falso	Condicion 3

Verdadero	Condicion 8

Condicion 3 - `isIdent(lexical1) && isIdent(lexical2)`	
Falso	Condicion 4
Verdadero	Condicion 9

Condicion 4 - `isIdent(lexical1) && isReg(lexical2)`	
Falso	Condicion 5
Verdadero	Condicion 10

Condicion 5 - `isReg(lexical1) && isIdent(lexical2)`	
Falso	Condicion 6
Verdadero	Condicion 11

Condicion 6 - `isReg(lexical1) && isIdent(lexical2)`	
Falso	Error semántico, la instrucción no tiene una estructura válida
Verdadero	Condicion 12

Condicion 7 - `lexical1.equals(lexical2)`	
Falso	Error semántico, los registros no son del mismo tamaño
Verdadero	Se agrega el registro a la hash table

Condicion 8 - `lexical2.equals("REG_8") && Integer.parseInt(val1)>255`

Falso	Se agregan los registros a la hashtable
Verdadero	Error semántico, el registro de 8 bits excede el tamaño límite

Condicion 9 - `existIdentifier(struct, val1) && existIdentifier(struct, val2)`	
Falso	Error semántico, no se puede acceder a los identificadores
Verdadero	Obtener el identificador solicitado

Condicion 10 - `existIdentifier(struct, val1)`	
Falso	Error semántico, no se puede acceder a la variable
Verdadero	Condicion 13

Condicion 11 - `existIdentifier(struct, val2)`	
Falso	Error semántico, no se puede acceder a la variable
Verdadero	Se obtiene la variable solicitada

Condicion 12 - `existIdentifier(struct, val2)`	
Falso	Error semántico, no se puede acceder a la variable
Verdadero	Se almacena el valor 1 en la variable solicitada

Condicion 13 - `lexical2.equals("REG_8") && Integer.parseInt(getVar(struct, val1).saved)>255`	
Falso	Almacenar en la hashtable la instrucción
Verdadero	Error semántico, el valor excede la capacidad

```
public boolean isReg(String value){
        if(value.equals("REG_16") || value.equals("REG_8")){
            return true;
        }
        return false;
    }
```

Condicion 1 - value.equals("REG_16") || value.equals("REG_8")	
Falso	El valor enviado no es un registro
Verdadero	El valor enviado es un registro de 8 o 16 bits

```
public boolean isNum(String value){
        if(value.equals("NUMERO")){
            return true;
        }
        return false;
    }
```

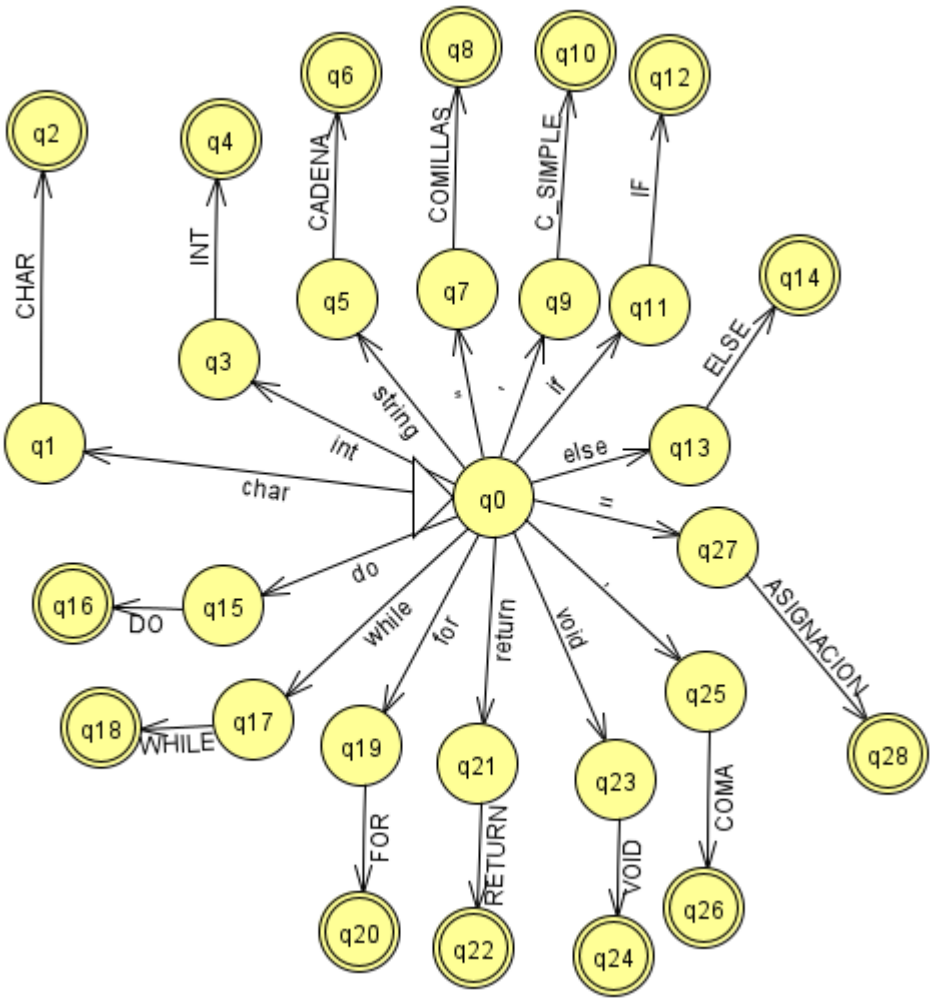
Condicion 1 - value.equals("NUMERO")	
Falso	El valor enviado no es un número
Verdadero	El valor enviado es un número

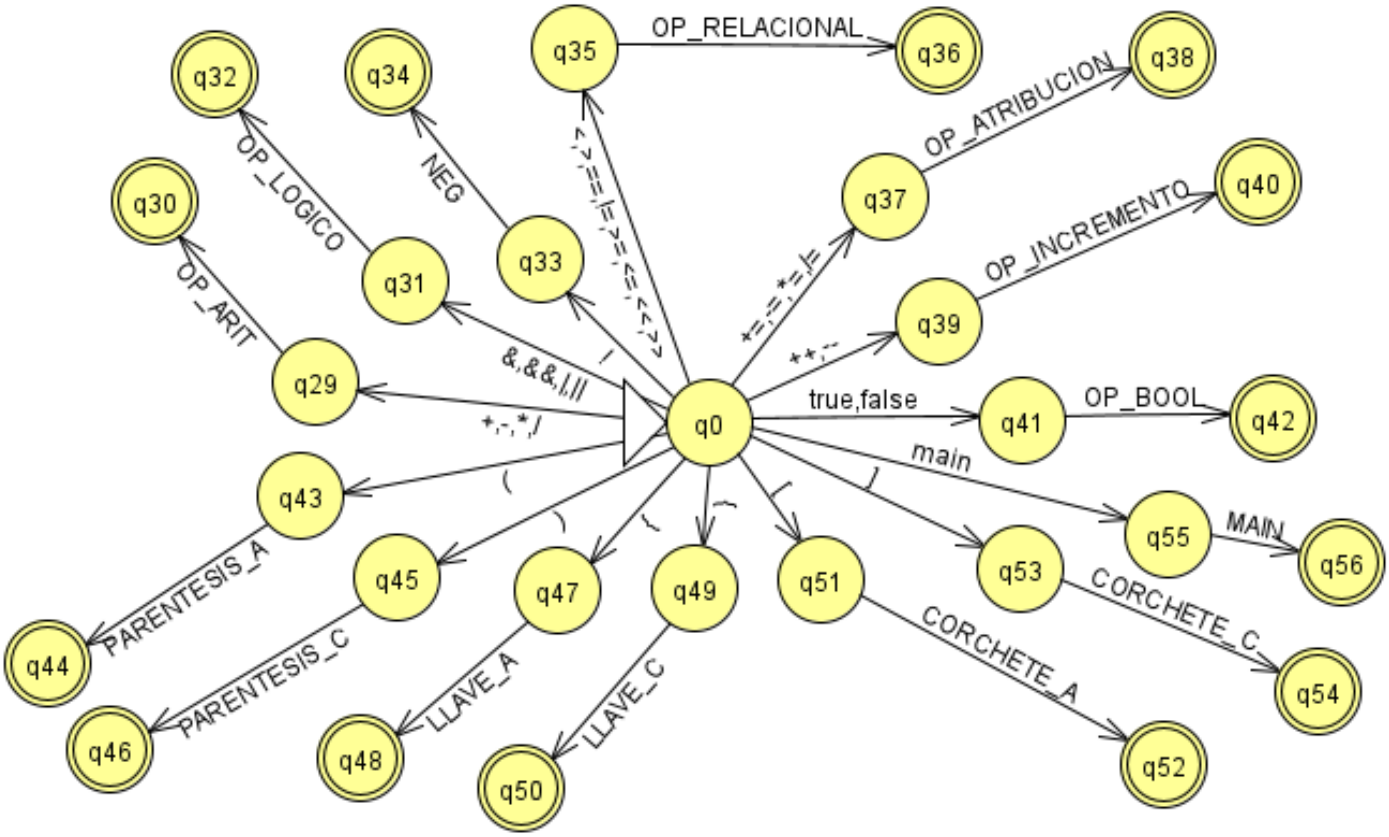
```
public boolean isIdent(String value){
        if(value.equals("IDENTIFICADOR")){
            return true;
        }
        return false;
    }
```

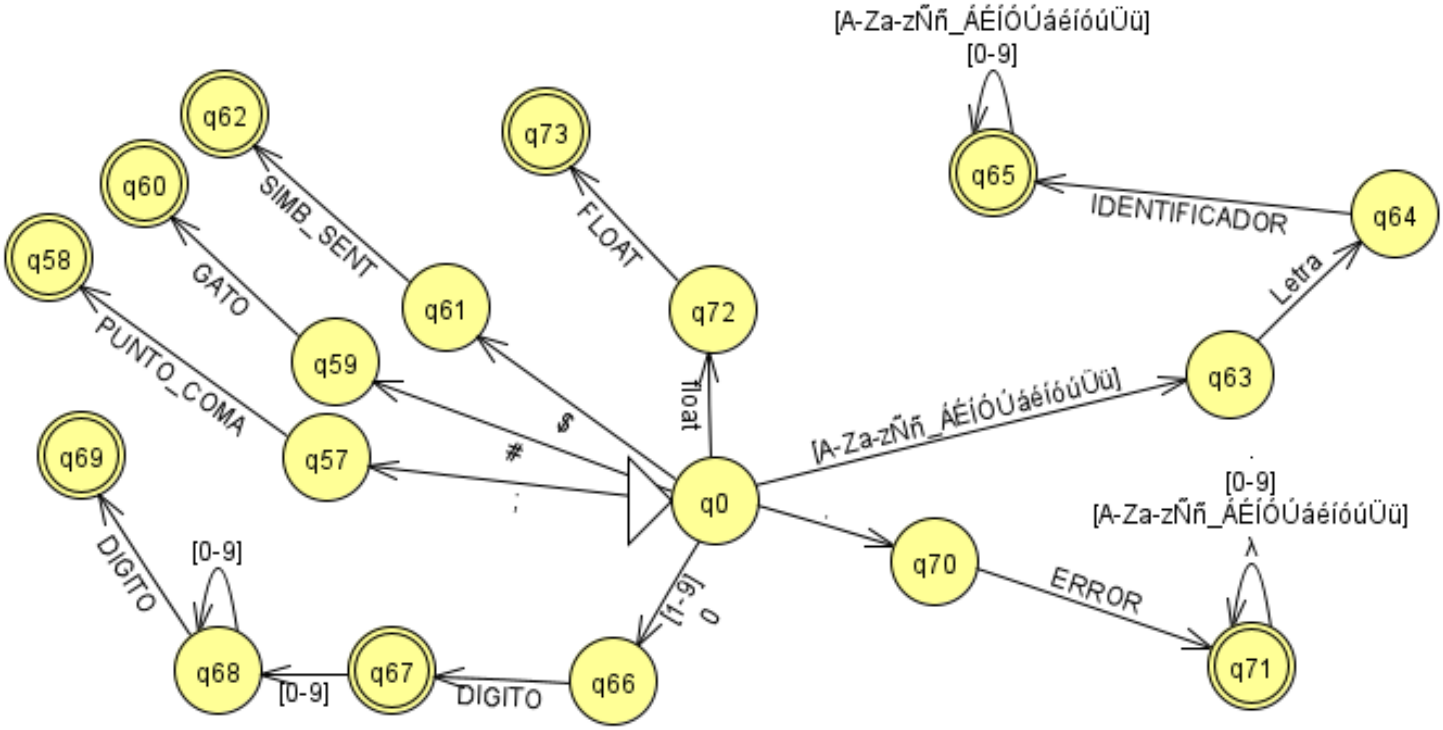
Condicion 1 - value.equals("IDENTIFICADOR")	
Falso	El valor enviado no es un identificador
Verdadero	El valor enviado es un identificador

Grafos

Para la elaboración del grafo se tomaron como entradas todos aquellos símbolos o identificadores que reconoce el programa, y como salidas el token que le corresponde al símbolo.
También cabe aclarar que los cuatro grafos de abajo son en realidad uno solo, pero los tuvimos que dividir para que fueran más legibles al momento de pegarlos en este documento.
En la tercera imagen, hay una flecha que va del estado 0 al 70, esta flecha tiene un punto, y ese punto representa todos aquellos símbolos o palabras que no reconoce nuestro programa, es por eso que manda el token de error.




q0
q29
q30
q31
q32
q33
q34
q35
q36
q37
q38
q39
q40
q41
q42
q43
q44
q45
q46
q47
q48
q49
q50
q51
q52
q53
q54
q55
q56
OP_ARIT
OP_LOGICO
NEG
OP_RELACIONAL
OP_ATRIBUCION
OP_INCREMENTO
OP_BOOL
true,false
main
MAIN
PARENTESIS_A
PARENTESIS_C
LLAVE_A
LLAVE_C
CORCHETE_A
CORCHETE_C
&,&&,|,||
+,-,*,/
(
)
{
}
[
]
++,--
+=,-=,*=,/=
<,>,==,!=,>=,<=


q0
q57
q58
q59
q60
q61
q62
q63
q64
q65
q66
q67
q68
q69
q70
q71
q72
q73
PUNTO_COMA
GATO
SIMB_SENT
FLOAT
IDENTIFICADOR
Letra
DIGITO
ERROR
float
$
#
;
.
[A-Za-zÑñ_ÁÉÍÓÚáéíóúÜü]
[0-9]
[1-9]
0
λ

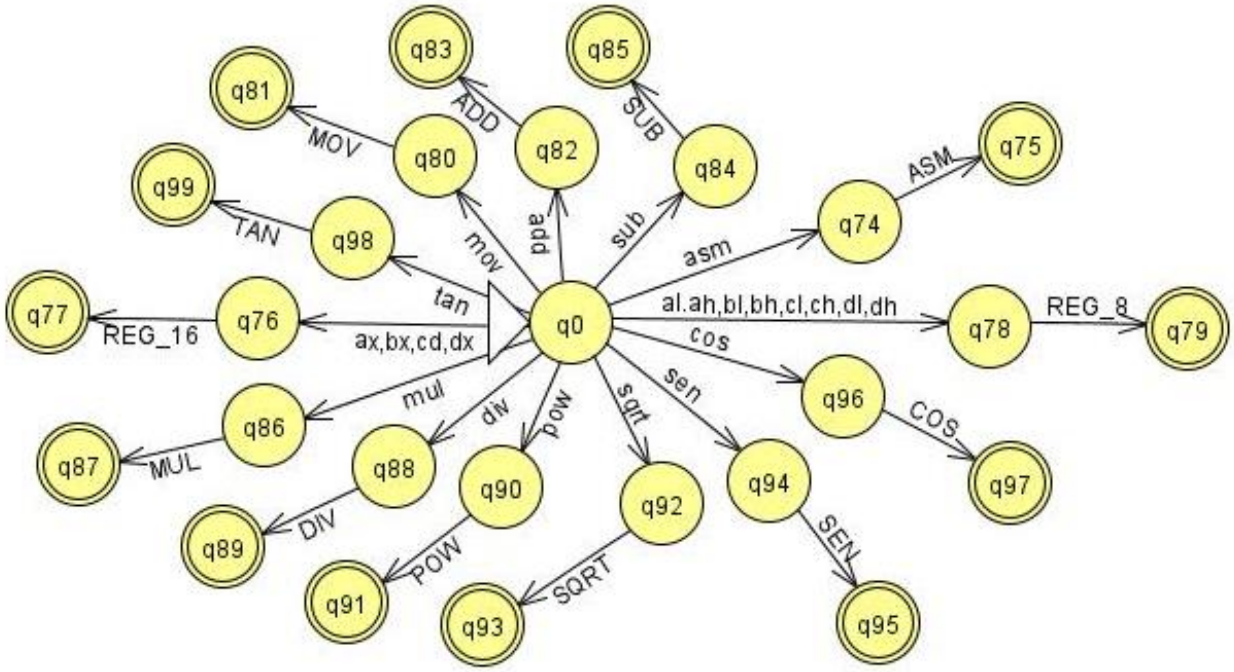


Tabla de transiciones

La primera columna representa los estados, y la primera fila los símbolos de entrada.

Todos aquellos estados o símbolos que no estén en las tablas se omitieron por practicidad, ya que eran casos en los que no había un cambio de estado sin que fueran lazos, por lo que no afectan en nada y hubieran hecho que la tabla creciera demasiado.

Los símbolos se agruparon de la siguiente manera para evitar que la tabla creciera mucho:

+={+,-,/,*}
& = {&,|,||,&&}
< = {<,>,<<,>>,==,!=,,<=,>=}
+= = {+=,-=,/=,*=}
++ = {++,--}
true = {true,false}
A = {[A-Z],[a-z],Ñ,ñ,_,Á,É,Í,Ó,Ú,á,é,í,ó,ú,Ü,ü}
ax = {ax,bx,cd,dx}
al = {al,ah,bl,bh,cl,ch,dl,dh}

	char	int	string	"		if	else	do	while	for	return
q0	q1	q3	q5	q7	q9	q11	q13	q15	q17	q19	q21

	void	,	=	=+	&	!	<	+=	++	true	(
q0	q23	q25	q27	q29	q31	q33	q35	q37	q39	q41	q43

	)	{	}	[	]	main	#	$	A	0	[1-9]	.	float	λ
q0	q45	q47	q49	q51	q53	q55	q57	q59	q61	q63	q66	q70	q72	N/A

	ax	tan	mov	add	sub	asm	al	cos	sen	sqrt	pow	div	mul
q0	q76	q98	q80	q82	q84	q74	q78	q96	q94	q92	q90	q88	q86

	λ
q1	q2
q3	q4
q5	q6
q7	q8
q9	q10
q11	q12
q13	q14
q15	q16
q17	q18
q19	q20
q21	q22
q23	q24
q25	q26
q27	q28
q29	q30
q31	q32
q33	q34
q35	q36
q37	q38
q39	q40

q41	q42
q43	q44
q45	q46
q47	q48
q49	q50
q51	q52
q53	q54
q55	q56
q57	q58
q59	q60
q61	q62
q63	q64
q64	q65
q74	q75
q76	q77
q78	q79
q80	q81
q82	q83
q84	q85
q86	q87
q88	q89
q90	q91
q92	q93
q94	q95
q96	q97
q98	q99

	A	0	[1-9]	.	λ
q65	q65	q65	q65	N/A	N/A
q66	N/A	N/A	N/A	N/A	q67
q67	N/A	q68	q68	N/A	N/A

q68	N/A	q68	q68	N/A	q69
q70	N/A	N/A	N/A	N/A	q71
q71	q71	q71	q71	q71	q71
q72	N/A	N/A	N/A	N/A	q73

Reglas de producción

Funciones correctas.
<DEC_FUNCION> → GATO VOID IDENTIFICADOR
<DEC_FUNCION_RET> → GATO (INT,FLOAT,CHAR,CADENA) IDENTIFICADOR

Incorrectas.
<DEC_FUNCION> → VOID IDENTIFICADOR
<DEC_FUNCION> → GATO VOID
<DEC_FUNCION> → GATO IDENTIFICADOR

Parámetros correctos.
<PARAMETROS> → SIMB_SENT (INT,FLOAT,CHAR,CADENA) IDENTIFICADOR

Declaraciones correctas.

<DECLARACION> → INT IDENTIFICADOR ASIGNACIÓN (IDENTIFICADOR | NUMERO) (OPARIT (IDENTIFICADOR | NUMERO))* PUNTO_COMA
<DECLARACIÓN>→ INT IDENTIFICADOR PUNTO_COMA
<DECLARACION> → FLOAT IDENTIFICADOR ASIGNACION (IDENTIFICADOR | NUMERO) OP_ARIT (IDENTIFICADOR | NUMERO) PUNTO_COMA
<DECLARACION> → FLOAT IDENTIFICADOR ASIGNACION (IDENTIFICADOR | NUMERO) PUNTO_COMA
<DECLARACION> → FLOAT IDENTIFICADOR PUNTO_COMA
<DECLARACION> → CHAR IDENTIFICADOR ASIGNACION C_SIMPLE (IDENTIFICADOR|NUMERO) C_SIMPLE PUNTO_COMA
<DECLARACION> → CHAR IDENTIFICADOR ASIGNACION IDENTIFICADOR PUNTO_COMA
<DECLARACION> → CHAR IDENTIFICADOR ASIGNACION C_SIMPLE C_SIMPLE PUNTO_COMA
<DECLARACION> → CHAR IDENTIFICADOR PUNTO_COMA
<DECLARACIÓN> → CADENA IDENTIFICADOR ASIGNACIÓN (IDENTIFICADOR | (COMILLAS IDENTIFICADOR COMILLAS)) (OP ARIT (IDENTIFICADOR | (COMILLAS IDENTIFICADOR COMILLAS)))* PUNTO_COMA
<DECLARACIÓN>→ CADENA IDENTIFICADOR PUNTO_COMA
<DECLARACIÓN>CADENA IDENTIFICADOR ASIGNACIÓN COMILLAS COMILLAS PUNTO COMA
<DECLARACIÓN> → CADENA IDENTIFICADOR ASIGNACIÓN COMILLAS IDENTIFICADOR COMILLAS PUNTO COMA

<DECLARACIÓN> → CADENA IDENTIFICADOR ASIGNACIÓN COMILLAS IDENTIFICADOR NUMERO PUNTO_COMA
<DECLARACIÓN> → IDENTIFICADOR ASIGNACIÓN (IDENTIFICADOR | NUMERO | (COMILLAS IDENTIFICADOR COMILLAS) | (C SIMPLE IDENTIFICADOR C SIMPLE)) PUNTO COMA
<DECLARACIÓN> → IDENTIFICADOR ASIGNACIÓN (NUMERO | IDENTIFICADOR | (COMILLAS IDENTIFICADOR COMILLAS) | (C SIMPLE IDENTIFICADOR C SIMPLE))(OPARIT (NUMERO | IDENTIFICADOR | (COMILLAS IDENTIFICADOR COMILLAS) | (C SIMPLE IDENTIFICADOR C SIMPLE)))* PUNTO_COMA

Incorrectas.

<DECLARACION> → INT IDENTIFICADOR ASIGNACION (IDENTIFICADOR | NUMERO) (IDENTIFICADOR | NUMERO)* PUNTO_COMA
<DECLARACION> → INT IDENTIFICADOR ASIGNACIÓN (IDENTIFICADOR | NUMERO) OPARIT PUNTO COMA
<DECLARACION> → INT IDENTIFICADOR ASIGNACIÓN (IDENTIFICADOR | NUMERO) (OPARIT (IDENTIFICADOR | NUMERO))*
<DECLARACIÓN> → INT IDENTIFICADOR PARÉNTESIS A
<DECLARACIÓN> → INT PUNTO_COMA
<DECLARACIÓN> → INT ASIGNACIÓN NUMERO PUNTO_COMA
<DECLARACIÓN> → INT IDENTIFICADOR ASIGNACIÓN NUMERO
<DECLARACIÓN> → INT IDENTIFICADOR
<DECLARACIÓN> → INT
<DECLARACION> → INT IDENTIFICADOR ASIGNACION (C_SIMPLE|COMILLAS) (IDENTIFICADOR | NUMERO) (C_SIMPLE|COMILLAS) OP_ARIT (((C_SIMPLE|COMILLAS) (IDENTIFICADOR | NUMERO) (C_SIMPLE|COMILLAS))|((IDENTIFICADOR | NUMERO))) PUNTO_COMA
<DECLARACION> → INT IDENTIFICADOR ASIGNACION (IDENTIFICADOR | NUMERO) OP_ARIT (C_SIMPLE|COMILLAS) (IDENTIFICADOR | NUMERO) (C_SIMPLE|COMILLAS) PUNTO_COMA
<DECLARACION> → INT IDENTIFICADOR ASIGNACION (C_SIMPLE|COMILLAS) (IDENTIFICADOR | NUMERO) (C_SIMPLE|COMILLAS) PUNTO_COMA
<DECLARACION> → FLOAT IDENTIFICADOR ASIGNACION (IDENTIFICADOR | NUMERO) (IDENTIFICADOR | NUMERO)* PUNTO_COMA
<DECLARACION> → FLOAT IDENTIFICADOR ASIGNACIÓN (IDENTIFICADOR | NUMERO) OPARIT PUNTO COMA
<DECLARACION> → FLOAT IDENTIFICADOR ASIGNACIÓN (IDENTIFICADOR | NUMERO) (OPARIT (IDENTIFICADOR | NUMERO))*
<DECLARACIÓN> → FLOAT IDENTIFICADOR PARÉNTESIS A
<DECLARACIÓN> → FLOAT PUNTO_COMA
<DECLARACIÓN> → FLOAT ASIGNACIÓN NUMERO PUNTO_COMA
<DECLARACIÓN> → FLOAT IDENTIFICADOR ASIGNACIÓN NUMERO
<DECLARACIÓN> → FLOAT IDENTIFICADOR
<DECLARACIÓN> → FLOAT
<DECLARACION> → FLOAT IDENTIFICADOR ASIGNACION (C_SIMPLE|COMILLAS) (IDENTIFICADOR | NUMERO) (C_SIMPLE|COMILLAS) OP_ARIT

(((C_SIMPLE|COMILLAS) (IDENTIFICADOR | NUMERO)
(C_SIMPLE|COMILLAS))|((IDENTIFICADOR | NUMERO))) PUNTO_COMA
<DECLARACION> → FLOAT IDENTIFICADOR ASIGNACION (IDENTIFICADOR | NUMERO) OP_ARIT (C_SIMPLE|COMILLAS) (IDENTIFICADOR | NUMERO) (C_SIMPLE|COMILLAS) PUNTO_COMA
<DECLARACION> → FLOAT IDENTIFICADOR ASIGNACION (C_SIMPLE|COMILLAS) (IDENTIFICADOR | NUMERO) (C_SIMPLE|COMILLAS) PUNTO_COMA
<DECLARACION> → CHAR IDENTIFICADOR ASIGNACION (IDENTIFICADOR | (C_SIMPLE IDENTIFICADOR C_SIMPLE)) (OP_ARIT (IDENTIFICADOR | (C_SIMPLE IDENTIFICADOR C_SIMPLE)))* PUNTO_COMA
<DECLARACION> → CHAR ASIGNACION C_SIMPLE (IDENTIFICADOR | NUMERO) C_SIMPLE PUNTO_COMA
<DECLARACION> → CHAR IDENTIFICADOR ASIGNACION C_SIMPLE (IDENTIFICADOR | NUMERO) C_SIMPLE
<DECLARACION> → CHAR IDENTIFICADOR ASIGNACION C_SIMPLE (IDENTIFICADOR | NUMERO) PUNTO_COMA
<DECLARACION> → CHAR IDENTIFICADOR ASIGNACION (IDENTIFICADOR | NUMERO) C_SIMPLE PUNTO_COMA
<DECLARACION> → CHAR IDENTIFICADOR ASIGNACION COMILLAS (IDENTIFICADOR|NUMERO) COMILLAS PUNTO_COMA
<DECLARACION> → CHAR IDENTIFICADOR ASIGNACION NUMERO PUNTO_COMA
<DECLARACION> → CHAR IDENTIFICADOR PARENTESIS_A
<DECLARACION> → CHAR PUNTO_COMA
<DECLARACION> → CHAR
<DECLARACIÓN> →CADENA IDENTIFICADOR ASIGNACIÓN (IDENTIFICADOR | (COMILLAS IDENTIFICADOR COMILLAS)) (IDENTIFICADOR | (COMILLAS IDENTIFICADOR COMILLAS))* PUNTO_COMA
<DECLARACIÓN> →CADENA IDENTIFICADOR ASIGNACIÓN (IDENTIFICADOR | (COMILLAS IDENTIFICADOR COMILLAS)) OP ARIT PUNTO COMA
<DECLARACIÓN> →CADENA IDENTIFICADOR ASIGNACIÓN (IDENTIFICADOR | (COMILLAS IDENTIFICADOR COMILLAS)) (OP ARIT (IDENTIFICADOR | (COMILLAS IDENTIFICADOR COMILLAS)))*
<DECLARACIÓN> →CADENA PUNTO_COMA
<DECLARACIÓN> →CADENA ASIGNACIÓN COMILLAS (IDENTIFICADOR | NUMERO) COMILLAS PUNTO COMA
<DECLARACIÓN> →CADENA IDENTIFICADOR COMILLAS (IDENTIFICADOR | NUMERO) COMILLAS
<DECLARACIÓN> →CADENA IDENTIFICADOR ASIGNACIÓN (IDENTIFICADOR | NUMERO) PUNTO_COMA
<DECLARACIÓN> → CADENA IDENTIFICADOR ASIGNACIÓN COMILLAS (IDENTIFICADOR | NUMERO) PUNTO_COMA
<DECLARACIÓN> →CADENA IDENTIFICADOR ASIGNACIÓN (IDENTIFICADOR | NUMERO) COMILLAS PUNTO COMA
<DECLARACIÓN> →CADENA
<DECLARACION CADENA IDENTIFICADOR ASIGNACION (IDENTIFICADOR | (COMILLAS IDENTIFICADOR COMILLAS)) OP_ARIT (C_SIMPLE IDENTIFICADOR C_SIMPLE) PUNTO_COMA

<DECLARACION> → CADENA IDENTIFICADOR ASIGNACION C_SIMPLE IDENTIFICADOR C_SIMPLE OP_ARIT (IDENTIFICADOR | (COMILLAS IDENTIFICADOR COMILLAS)) PUNTO_COMA
<DECLARACION> → CADENA IDENTIFICADOR ASIGNACION C_SIMPLE IDENTIFICADOR C_SIMPLE OP_ARIT C_SIMPLE IDENTIFICADOR C_SIMPLE PUNTO_COMA
<DECLARACIÓN> →IDENTIFICADOR ASIGNACIÓN (NUMERO | IDENTIFICADOR | (COMILLAS IDENTIFICADOR COMILLAS) | (C SIMPLE IDENTIFICADOR C SIMPLE)) (NUMERO | IDENTIFICADOR | (COMILLAS IDENTIFICADOR COMILLAS) | (C SIMPLE IDENTIFICADOR C SIMPLE))* PUNTO_COMA
<DECLARACIÓN> →IDENTIFICADOR ASIGNACIÓN (NUMERO | IDENTIFICADOR | (COMILLAS IDENTIFICADOR COMILLAS) | (C SIMPLE IDENTIFICADOR C SIMPLE)) OPARIT PUNTO COMA
<DECLARACIÓN> →IDENTIFICADOR ASIGNACIÓN (NUMERO | IDENTIFICADOR | (COMILLAS IDENTIFICADOR COMILLAS) | (C SIMPLE IDENTIFICADOR C SIMPLE)) (OPARIT (NUMERO | IDENTIFICADOR | (COMILLAS IDENTIFICADOR COMILLAS) | (C SIMPLE IDENTIFICADOR C SIMPLE)))*
<DECLARACIÓN> →IDENTIFICADOR IDENTIFICADOR PUNTO_COMA
<DECLARACIÓN> →IDENTIFICADOR IDENTIFICADOR
<DECLARACIÓN> →IDENTIFICADOR ASIGNACIÓN (IDENTIFICADOR | NUMERO | (COMILLAS IDENTIFICADOR COMILLAS) | (C SIMPLE IDENTIFICADOR C SIMPLE))
<DECLARACION> →ASIGNACIÓN (IDENTIFICADOR | NUMERO | (COMILLAS IDENTIFICADOR COMILLAS) | (C SIMPLE IDENTIFICADOR C SIMPLE)) PUNTO COMA
<DECLARACIÓN> →IDENTIFICADOR ASIGNACIÓN PUNTO_COMA
<DECLARACIÓN> →IDENTIFICADOR (IDENTIFICADOR | NUMERO | (COMILLAS IDENTIFICADOR COMILLAS) | (C SIMPLE IDENTIFICADOR C SIMPLE)) PUNTO COMA
<DECLARACIÓN> →OP INCREMENTÓ PUNTO COMA
<DECLARACIÓN> →IDENTIFICADOR OP ATRIBUCIÓN PUNTO COMA
<DECLARACIÓN> →IDENTIFICADOR PUNTO_COMA
<DECLARACIÓN> →OP ATRIBUCIÓN PUNTO COMA

Comparación correcta.
<BOLEANA> → SIMB_SENT (NEG)? IDENTIFICADOR OP RELACIONAL (OP BOOL | NUMERO | IDENTIFICADOR| COMILLAS COMILLAS | COMILLAS (IDENTIFICADOR | NUMERO) COMILLAS)(OP_LOGICO SIMB_SENT (NEG)? IDENTIFICADOR OP RELACIONAL (OP BOOL | NUMERO | IDENTIFICADOR | COMILLAS COMILLAS | COMILLAS (IDENTIFICADOR | NUMERO) COMILLAS))+
<BOLEANA>→ SIMB_SENT (NEG)? IDENTIFICADOR OP RELACIONAL (OP BOOL | NUMERO | IDENTIFICADOR | COMILLAS COMILLAS | COMILLAS (IDENTIFICADOR | NUMERO) COMILLAS)

Incorrectas.
<BOOLEANA>→ SIMB_SENT IDENTIFICADOR OP RELACIONAL
<BOLEANA>→ SIMB_SENT (NEG)? IDENTIFICADOR OP RELACIONAL (OP BOOL | NUMERO | IDENTIFICADOR | COMILLAS COMILLAS | COMILLAS (IDENTIFICADOR | NUMERO) COMILLAS)

<BOLEANA>→ SIMB_SENT (NEG)? IDENTIFICADOR OP RELACIONAL (OP BOOL | NUMERO | IDENTIFICADOR | COMILLAS COMILLAS | COMILLAS (IDENTIFICADOR | NUMERO) COMILLAS)
<BOLEANA>→ OP BOOL
<BOLEANA> → OP_LOGICO

Return correctos.
<RETORNO> → RETURN (NUMERO | IDENTIFICADOR | OP_BOOL) PUNTO COMA

Incorrectas.
<RETORNO> →RETURN (NUMERO | IDENTIFICADOR | OP_BOOL)
<RETORNO> →RETURN

Estructuras correctas.
<DECL_FOR> →SIMB_SENT IDENTIFICADOR OP ATRIBUCIÓN NUMERO
<DECL_FOR> →SIMB_SENT IDENTIFICADOR OP INCREMENTO
<DECL_FOR> →SIMB_SENT OP INCREMENTO IDENTIFICADOR
<SENT IF ELSE> →IF PARENTESIS A <BOLEANA> PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C ELSE LLAVE A (<SENTENCIA>)* LLAVE C
<SENT IF> →IF PARENTESIS A <BOLEANA> PARENTESIS C LLAVE A (<SENTENCIA>* LLAVE C
<SENT_FOR> →<SENTENCIA> <BOLEANA> PUNTO COMA <DECL_FOR>
<FOR C> →FOR PARENTESIS A <SENT FOR> PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C
<WHILE C> →WHILE PARENTESIS A <BOLEANA> PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C
<SENTENCIA> →(<SENTENCIA> | <DECLARACION> | <SENT_IF> | <SENT_IFELSE> | <FOR_C> | <WHILE_C>)*
<SENT IF ELSE> →IF PARENTESIS A <BOLEANA> PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C ELSE LLAVE A (<SENTENCIA>)* LLAVE C
<SENT IF> →IF PARENTESIS A <BOLEANA> PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C
<SENT_FOR> →<SENTENCIA> <BOLEANA> PUNTO COMA <DECL_FOR>
<FOR C> →FOR PARENTESIS A <SENT FOR> PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C
<WHILE C> →WHILE PARENTESIS A <BOLEANA> PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C
<SENTENCIA> →(<SENTENCIA> | <DECLARACION> | <SENT_IF> | <SENT_IFELSE> | <FOR_C> | <WHILE_C>)*
<SENT IF> → IF PARENTESIS A <BOLEANA> PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C
<SENT_FOR> → <SENTENCIA> <BOLEANA> PUNTO COMA <DECL_FOR>
<WHILE C> →WHILE PARENTESIS A <BOLEANA> PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C
<SENTENCIA> →(<SENTENCIA> | <DECLARACION> | <SENT_IF> | <SENT_IFELSE> | <FOR_C> | <WHILE_C>)*
<FOR C> →FOR PARENTESIS A <SENT FOR> PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C

<SENTENCIA> →(<SENTENCIA> | <DECLARACION> | <SENT_IF> | <SENT_IFELSE> | <FOR_C> | <WHILE_C>)*
<WHILE C> → WHILE PARENTESIS A <BOLEANA> PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C

Incorrectas.
<DECL_FOR> →IDENTIFICADOR OP ATRIBUCIÓN NUMERO
<DECL_FOR> →IDENTIFICADOR OP INCREMENTO
<DECL_FOR> →OP INCREMENTO IDENTIFICADOR
<DECL_FOR> →SIMB_SENT IDENTIFICADOR OP ATRIBUCIÓN
<DECL_FOR> →SIMB_SENT IDENTIFICADOR NUMERO
<DECL_FOR> →SIMB_SENT OP_ATRIBUCIONNUMERO
<DECL_FOR> →SIMB_SENT IDENTIFICADOR
<DECL_FOR> →SIMB_SENT OP ATRIBUCIÓN NUMERO
<DECL_FOR> →SIMB_SENT OP INCREMENTO
<SENT IF ELSE> →IF PARENTESIS A <BOLEANA> PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C ELSE (<SENTENCIA>)* LLAVE C
<SENT IF ELSE> →IF PARENTESIS A <BOLEANA> PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C ELSE LLAVE A <(SENTENCIA>)*
<SENT_IFELSE> →IF PARENTESIS A PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C ELSE LLAVE A (<SENTENCIA>)* LLAVE C
<SENT_IFELSE> →IF PARENTESIS A <BOLEANA> PARENTESIS_C (<SENTENCIA>)* LLAVE C ELSE LLAVE A (<SENTENCIA>)* LLAVE C
<SENT IF ELSE> →IF PARENTESIS A <BOLEANA> PARENTESIS C LLAVE A (<SENTENCIA>)* ELSE LLAVE A (<SENTENCIA>)* LLAVE C
<SENT_IFELSE> →IF <BOLEANA> PARENTESIS_C LLAVE A (<SENTENCIA>)* LLAVE C ELSE LLAVE A (<SENTENCIA)* LLAVE C
<SENT_IFELSE> →IF PARENTESIS A <BOLEANA> LLAVE A (<SENTENCIA>)* LLAVE C ELSE LLAVE A (<SENTENCIA>)* LLAVE C
<SENT IF> →IF PARENTESIS_A PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C
<SENT IF> →IF <BOLEANA> PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C
<SENT IF> →IF PARENTESIS A <BOLEANA> LLAVE A (<SENTENCIA>)* LLAVE C
<SENT IF> →IF PARENTESIS A <BOLEANA> PARENTESIS_C (<SENTENCIA>)* LLAVE C
<SENT IF> →IF PARENTESIS A <BOLEANA> PARENTESIS C LLAVE A (<SENTENCIA>)
<SENT IF> →IF
<SENT IF> →ELSE
<SENT_FOR> →<BOLEANA> PUNTO COMA <DECL_FOR>
<SENT_FOR> →<SENTENCIA> PUNTO COMA <DECL_FOR>
<SENT_FOR> →<SENTENCIA> <BOLEANA> <DECL_FOR>
<SENT_FOR> →<SENTENCIA> <BOLEANA> PUNTO COMA
<SENT_FOR> →<SENTENCIA> <BOLEANA> PUNTO COMA
<FOR C> →FOR <SENT FOR> PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C
<FOR C> →FOR PARÉNTESIS A <SENT_FOR> LLAVE A (<SENTENCIA>)* LLAVE C
<FOR_C> →FOR PARENTESIS_A <SENT_FOR> PARENTESIS_C (<SENTENCIA>)* LLAVE C
<FOR C> →FOR PARENTESIS A <SENT FOR> PARENTESIS C LLAVE A (<SENTENCIA>)*

<FOR C> →FOR PARENTESIS_A PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C
<FOR C> →FOR
<WHILE C> →WHILE <BOLEANA> PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C
<WHILE C> →WHILE PARÉNTESIS A <BOLEANA> LLAVE A (<SENTENCIA>)* LLAVE C
<WHILE C> →WHILE PARÉNTESIS A <BOLEANA> PARENTESIS_C (<SENTENCIA>)* LLAVE C
<WHILE C> →WHILE PARENTESIS A <BOLEANA> PARENTESIS C LLAVE A (<SENTENCIA>)*
<WHILE C> →WHILE PARENTESIS_A PARENTESIS_C LLAVE_A (<SENTENCIA>)* LLAVE C
<WHILE C> →WHILE

Funciones correctas.
<FUNCION> →<DEC_FUNCION_RET> PARENTESIS A (<PARAMETROS> (COMA PARAMETROS)+)* PARENTESIS_C LLAVE A (<SENTENCIA>)* <RETORNO> LLAVE C
<FUNCION> →<DEC_FUNCION_RET> PARENTESIS A (<PARAMETROS>)* PARENTESIS_C LLAVE A (<SENTENCIA>)*< RETORNO> LLAVE C
<FUNCION> →<DEC_FUNCION> PARENTESIS A (<PARAMETROS> (COMA <PARAMETROS>)+)* PARENTESIS_C LLAVE A (<SENTENCIA>)* LLAVE C
<FUNCION> →<DEC_FUNCION> PARENTESIS A (<PARAMETROS>)* PARENTESIS_C LLAVE A (<SENTENCIA>)* LLAVE C

Incorrectas.
<FUNCION> →<DEC_FUNCION_RET> PARENTESIS A (<PARAMETROS> (COMA <PARAMETROS>)+)* PARENTESIS_C LLAVE A (<SENTENCIA>)* LLAVE C
<FUNCION> →<DEC_FUNCION_RET> (<PARAMETROS> (COMA <PARAMETROS>)+)* PARENTESIS_C LLAVE A (<SENTENCIA>)* RETORNO LLAVE C
<FUNCION> →<DEC_FUNCION_RET> PARENTESIS A (<PARAMETROS> (COMA <PARAMETROS>)+)* LLAVE A (<SENTENCIA>)* <RETORNO> LLAVE C
<FUNCION> →<DEC_FUNCION_RET> PARENTESIS A (<PARAMETROS> (COMA <PARAMETROS>)+)* PARENTESIS_C (<SENTENCIA>)* <RETORNO> LLAVE C
<FUNCION> →<DEC_FUNCION_RET> PARENTESIS A (<PARAMETROS> (COMA <PARAMETROS>)+)* PARENTESIS_C LLAVE A (<SENTENCIA>)* <RETORNO>
<FUNCION>→<DEC_FUNCION_RET> PARENTESIS A (<PARAMETROS> (<PARAMETROS>)+)* PARENTESIS_C LLAVE A (<SENTENCIA>)* <RETORNO> LLAVE C
<FUNCION> →<DEC_FUNCION_RET> PARENTESIS A (<PARAMETROS>)* PARENTESIS_C LLAVE A (<SENTENCIA>)* LLAVE C
<FUNCION> →<DEC_FUNCION_RET> (<PARAMETROS>)* PARENTESIS_C LLAVE A (<SENTENCIA>)* <RETORNO> LLAVE C
<FUNCION> →<DEC_FUNCION_RET> PARENTESIS A (<PARAMETROS>)*LLAVE A (<SENTENCIA>)* <RETORNO> LLAVE C
<FUNCION> →<DEC_FUNCION_RET> PARENTESIS A (<PARAMETROS>)* PARENTESIS_C (<SENTENCIA>)* <RETORNO> LLAVE C
<FUNCION> →<DEC_FUNCION_RET> PARENTESIS A (<PARAMETROS>)* PARENTESIS_C LLAVE A (<SENTENCIA>)* <RETORNO>
<FUNCION> →<DEC_FUNCION> (<PARAMETROS> (COMA <PARAMETROS>)+)* PARENTESIS_C LLAVE A (<SENTENCIA>)* LLAVE C

<FUNCION> →<DEC_FUNCION> PARENTESIS A (<PARAMETROS> (COMA <PARAMETROS>)+)* LLAVE A (<SENTENCIA>)* LLAVE C
<FUNCION> →<DEC_FUNCION> PARENTESIS A (<PARAMETROS> (COMA <PARAMETROS>)+)* PARENTESIS_C (<SENTENCIA>)* LLAVE C
<FUNCION> →<DEC_FUNCION> PARENTESIS A (<PARAMETROS> (COMA <PARAMETROS>)+)* PARENTESIS_C LLAVE A (<SENTENCIA>)*
<FUNCION> →<DEC_FUNCION> PARENTESIS A (<PARAMETROS> (<PARAMETROS>)+)* PARENTESIS_C LLAVE A (<SENTENCIA>)* LLAVE C
<FUNCION> →<DEC_FUNCION> (<PARAMETROS>)* PARENTESIS_C LLAVE A (<SENTENCIA>)* LLAVE C
<FUNCION> →<DEC_FUNCION> PARENTESIS A (<PARAMETROS>)*LLAVE A (<SENTENCIA>)* LLAVE C
<FUNCION> →<DEC_FUNCION> PARENTESIS A (<PARAMETROS>)* PARENTESIS_C (<SENTENCIA>)* LLAVE C
<FUNCION> →<DEC_FUNCION> PARENTESIS A (<PARAMETROS>)* PARENTESIS_C LLAVE A (<SENTENCIA>)*

Metodo main correctas.
<PRINCIPAL> →MAIN PARENTESIS_A PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C

Incorrectas.
<PRINCIPAL> →MAIN PARENTESIS_C LLAVE A (<SENTENCIA>)* LLAVE C
<PRINCIPAL> →MAIN PARÉNTESIS A LLAVE A (<SENTENCIA>)* LLAVE C
<PRINCIPAL> →MAIN PARENTESIS_A PARENTESIS_C (<SENTENCIA>)* LLAVE C
<PRINCIPAL> →MAIN PARENTESIS_A PARENTESIS C LLAVE A (<SENTENCIA>)*
<PRINCIPAL> →MAIN PARENTESIS_C LLAVE A (<SENTENCIA>)* LLAVE C
<PRINCIPAL> →MAIN PARÉNTESIS A LLAVE A (<SENTENCIA>)* LLAVE C
<PRINCIPAL> →MAIN PARENTESIS_A PARENTESIS_C (<SENTENCIA>)* LLAVE C
<PRINCIPAL> →MAIN PARENTESIS_A PARENTESIS C LLAVE A (<SENTENCIA>)*
<PRINCIPAL> →(MAIN | IDENTIFICADOR) PARENTESIS_A PARENTESIS C LLAVE A (<SENTENCIA>)* LLAVE C
<PRINCIPAL> →MAIN

Sentencia correctas.
<SENTENCIA> →(<SENTENCIA> | <DECLARACION> | <SENT_IF> | <SENT_IFELSE> | <FOR_C> | <WHILE_C>)*

Return correctas.
<RETORNO> → RETURN (NUMERO | IDENTIFICADOR | OP_BOOL | (COMILLAS IDENTIFICADOR COMILLAS) | (COMILLAS COMILLAS) | (C_SIMPLE IDENTIFICADOR C_SIMPLE) | (C_SIMPLE C_SIMPLE)) PUNTO_COMA
<RETORNO> → RETURN PUNTO_COMA

Incorrectas.
<RETORNO> → RETURN (NUMERO | IDENTIFICADOR | OP_BOOL)
<RETORNO> → RETURN

Llamadas a función correctas.
<LLAMADA_FUNCT> → IDENTIFICADOR PARENTESIS_A ((NUMERO | IDENTIFICADOR | (COMILLAS (NUMERO | IDENTIFICADOR) COMILLAS) | (C_SIMPLE (NUMERO | IDENTIFICADOR) C_SIMPLE) | (COMILLAS COMILLAS) | (C_SIMPLE C_SIMPLE)) (COMA (NUMERO | IDENTIFICADOR | (COMILLAS (NUMERO | IDENTIFICADOR) COMILLAS) | (C_SIMPLE (NUMERO | IDENTIFICADOR) C_SIMPLE) | (COMILLAS COMILLAS) | (C_SIMPLE C_SIMPLE)))+)* PARENTESIS_C PUNTO_COMA

Incorrectas
<LLAMADA_FUNCT> → IDENTIFICADOR PARENTESIS_A ((NUMERO | IDENTIFICADOR | (COMILLAS (NUMERO | IDENTIFICADOR) COMILLAS) | (C_SIMPLE (NUMERO | IDENTIFICADOR) C_SIMPLE) | (COMILLAS COMILLAS) | (C_SIMPLE C_SIMPLE)) (COMA (NUMERO | IDENTIFICADOR | (COMILLAS (NUMERO | IDENTIFICADOR) COMILLAS) | (C_SIMPLE (NUMERO | IDENTIFICADOR) C_SIMPLE) | (COMILLAS COMILLAS) | (C_SIMPLE C_SIMPLE)))+)* PARENTESIS_C

<LLAMADA_FUNCT> → IDENTIFICADOR PARENTESIS_A ((NUMERO | IDENTIFICADOR | (COMILLAS (NUMERO | IDENTIFICADOR) COMILLAS) | (C_SIMPLE (NUMERO | IDENTIFICADOR) C_SIMPLE) | (COMILLAS COMILLAS) | (C_SIMPLE C_SIMPLE) ((COMA)? (NUMERO | IDENTIFICADOR | (COMILLAS (NUMERO | IDENTIFICADOR) COMILLAS) | (C_SIMPLE (NUMERO | IDENTIFICADOR) C_SIMPLE) | (COMILLAS COMILLAS) | (C_SIMPLE C_SIMPLE)))+)* PARENTESIS_C PUNTO_COMA

Print correctas.
"IMPRIMIR → PRINT PARENTESIS_A COMILLAS (((SENTENCIA)? | (IDENTIFICADOR)? | (NUMERO)? | (REG_16)? | (REG_8)? | (ASIGNACION)?)* COMILLAS (OP_ARIT IDENTIFICADOR (OP_ARIT)?)*)* PARENTESIS_C PUNTO_COMA
IMPRIMIR → PRINT PARENTESIS_A IDENTIFICADOR PARENTESIS_C PUNTO_COMA

Incorrectos.
IMPRIMIR → PRINT PARENTESIS_A COMILLAS (((SENTENCIA)? | (IDENTIFICADOR)? | (NUMERO)? | (REG_16)? | (REG_8)? | (ASIGNACION)?)* COMILLAS (OP_ARIT IDENTIFICADOR (OP_ARIT)?)*)* PARENTESIS_C
IMPRIMIR → PRINT COMILLAS (((SENTENCIA)? | (IDENTIFICADOR)? | (NUMERO)? | (REG_16)? | (REG_8)? | (ASIGNACION)?)* COMILLAS (OP_ARIT IDENTIFICADOR (OP_ARIT)?)*)* PARENTESIS_C PUNTO_COMA
IMPRIMIR → PRINT PARENTESIS_A COMILLAS ((((SENTENCIA)? | (IDENTIFICADOR)? | (NUMERO)? | (REG_16)? | (REG_8)? | (ASIGNACION)?)* COMILLAS (OP_ARIT IDENTIFICADOR (OP_ARIT)?)*)* PUNTO_COMA
IMPRIMIR → PRINT PARENTESIS_A (((SENTENCIA)? | (IDENTIFICADOR)? | (NUMERO)? | (REG_16)? | (REG_8)? | (ASIGNACION)?)* COMILLAS (OP_ARIT IDENTIFICADOR (OP_ARIT)?)*)* PARENTESIS_C PUNTO_COMA
IMPRIMIR → PRINT PARENTESIS_A COMILLAS (((SENTENCIA)? | (IDENTIFICADOR)? | (NUMERO)? | (REG_16)? | (REG_8)? | (ASIGNACION)?)* (OP_ARIT IDENTIFICADOR (OP_ARIT)?)*)* PARENTESIS_C PUNTO_COMA
IMPRIMIR –> PRINT PARENTESIS_A COMILLAS (((SENTENCIA)? | (IDENTIFICADOR)? | (NUMERO)? | (REG_16)? | (REG_8)? | (ASIGNACION)?)* COMILLAS (OP_ARIT (OP_ARIT)?)*)* PARENTESIS_C PUNTO_COMA

Sen, cos y tan correctos.
ENSAMBLADOR → "ASM PARENTESIS_A COMILLAS (SEN | COS | TAN) (NUMERO | IDENTIFICADOR | REG_16 | REG_8) COMA (REG_16 | REG_8 | IDENTIFICADOR) COMILLAS PARENTESIS_C PUNTO_COMA

Pow correctos.
ENSAMBLADOR → ASM PARENTESIS_A COMILLAS POW (NUMERO | IDENTIFICADOR | REG_16 | REG_8) COMA NUMERO COMILLAS PARENTESIS_C PUNTO_COMA
Mov, add y sub correctos.
ENSAMBLADOR → ASM PARENTESIS_A COMILLAS (MOV | ADD | SUB) (NUMERO | IDENTIFICADOR | REG_16 | REG_8) COMA (REG_16 | REG_8 | IDENTIFICADOR) COMILLAS PARENTESIS_C PUNTO_COMA

Incorrectos.
ENSAMBLADOR → ASM COMILLAS (MOV | ADD | SUB) (NUMERO | IDENTIFICADOR | REG_16 | REG_8) COMA (REG_16 | REG_8 | IDENTIFICADOR) COMILLAS PARENTESIS_C PUNTO_COMA
ENSAMBLADOR → ASM PARENTESIS_A COMILLAS (MOV | ADD | SUB) (NUMERO | IDENTIFICADOR | REG_16 | REG_8) COMA (REG_16 | REG_8 | IDENTIFICADOR) COMILLAS PARENTESIS_C
ENSAMBLADOR → ASM PARENTESIS_A COMILLAS (MOV | ADD | SUB) (NUMERO | IDENTIFICADOR | REG_16 | REG_8) COMA (REG_16 | REG_8 | IDENTIFICADOR) COMILLAS PUNTO_COMA
ENSAMBLADOR → ASM PARENTESIS_A (MOV | ADD | SUB) (NUMERO | IDENTIFICADOR | REG_16 | REG_8) COMA (REG_16 | REG_8 | IDENTIFICADOR) COMILLAS PARENTESIS_C PUNTO_COMA
ENSAMBLADOR → ASM PARENTESIS_A COMILLAS (MOV | ADD | SUB) (NUMERO | IDENTIFICADOR | REG_16 | REG_8) COMA (REG_16 | REG_8 | IDENTIFICADOR) PARENTESIS_C PUNTO_COMA
ENSAMBLADOR → ASM PARENTESIS_A COMILLAS (MOV | ADD | SUB) COMA (REG_16 | REG_8 | IDENTIFICADOR) COMILLAS PARENTESIS_C PUNTO_COMA
ENSAMBLADOR → ASM PARENTESIS_A COMILLAS (MOV | ADD | SUB) (NUMERO | IDENTIFICADOR | REG_16 | REG_8) COMA COMILLAS PARENTESIS_C PUNTO_COMA

Mul, div y sqrt correctos.
ENSAMBLADOR → ASM PARENTESIS_A COMILLAS (MUL|DIV|SQRT) (NUMERO | IDENTIFICADOR | REG_16 | REG_8) COMILLAS PARENTESIS_C PUNTO_COMA

Incorrectos.
ENSAMBLADOR → ASM COMILLAS (MUL|DIV|SQRT) (NUMERO | IDENTIFICADOR | REG_16 | REG_8) COMILLAS PARENTESIS_C PUNTO_COMA
ENSAMBLADOR → ASM PARENTESIS_A COMILLAS (MUL|DIV|SQRT) (NUMERO | IDENTIFICADOR | REG_16 | REG_8) COMILLAS PARENTESIS_C
ENSAMBLADOR → ASM PARENTESIS_A COMILLAS (MUL|DIV|SQRT) (NUMERO | IDENTIFICADOR | REG_16 | REG_8) COMILLAS PUNTO_COMA
ENSAMBLADOR → ASM PARENTESIS_A (MUL|DIV|SQRT) (NUMERO | IDENTIFICADOR | REG_16 | REG_8) COMILLAS PARENTESIS_C PUNTO_COMA

ENSAMBLADOR → ASM PARENTESIS_A COMILLAS (MUL|DIV|SQRT) (NUMERO | IDENTIFICADOR | REG_16 | REG_8) PARENTESIS_C PUNTO_COMA
ENSAMBLADOR → ASM PARENTESIS_A COMILLAS (MUL|DIV|SQRT) COMILLAS PARENTESIS_C PUNTO_COMA