Universidad de San Carlos de Guatemala

Facultad de Ingenieria

Organización de Lenguajes y Compiladores 1

Auxiliar: Daniel Acabal

Proyecto 1 DataForge

Manual Técnico

Helen Janet Rodas Castro

202200066

Primer Semestre

Guatemala 10 de marzo del 2024

Introducción

El desarrollo de software es un campo amplio y multidisciplinario que requiere de una sólida comprensión de los fundamentos teóricos y prácticos de la informática. En particular, la construcción de compiladores, que son herramientas fundamentales en la creación de software, involucra la aplicación de conocimientos sobre análisis léxico y sintáctico.

En el contexto del curso de Organización de Lenguajes y Compiladores 1 de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, se presenta la necesidad de crear un sistema capaz de realizar operaciones aritméticas y estadísticas, así como de generar diversos gráficos a partir de una colección de datos. Esto implica la aplicación práctica de los conceptos y técnicas aprendidos en el curso para diseñar y desarrollar un analizador léxico y sintáctico eficiente.

El objetivo principal de este proyecto es poner en práctica los conocimientos, así como fortalecer las habilidades en la construcción de software mediante la implementación de un sistema funcional y versátil. A través de este proyecto, se podrá profundizar en el proceso de desarrollo de compiladores y aplicar estos conocimientos en la creación de soluciones de software prácticas y útiles en diversos contextos.

En este sentido, el presente trabajo se centra en la aplicación de los principios de análisis léxico y sintáctico para la construcción de un sistema que cumpla con los requisitos mencionados, brindando una oportunidad invaluable para integrar la teoría con la práctica y adquirir experiencia en el desarrollo de software de calidad.

El programa está hecho con lenguaje java, se utilizan las librerías JFLEX y CUP para los análisis léxicos y sintácticos, también la librería JFREECHART para las gráficas.

Da inicio en el FrmPrincipal.java el cual contiene todo lo relacionado con la interfaz grafica y las acciones que llaman a cada uno de los métodos para que se puedan ejecutar debidamente.

```
JPanel panel = (JPanel) jTabbedPaneArchivos.getComponentAt(index);

// Obtener el JTextArea dentro del panel
Component[] components = panel.getComponents();
for (Component component : components) {
    if (component instanceof JScrollPane) {
        JScrollPane scrollPane = (JScrollPane) component;
        Component viewportView = scrollPane.getViewport().getView();
        if (viewportView instanceof JTextArea) {
            JTextArea textArea = (JTextArea) viewportView;

            // Obtener el contenido del JTextArea
            String content = textArea.getText();

            LexerCup scan = new LexerCup(new StringReader( s:content));
            Parser parser = new Parser( s:scan);
            try {
                  parser.parse();
                arbol raiz = (arbol)parser.parse().value;
                raiz.run(raiz,TablaSim, areaConsola:TextAreaConsola);

                System.out.println( x:"---Tabla de simbolos---");
                for (CTablaSimb elemento : TablaSim ) {
                    System.out.println("Rol: " + elemento.rol + "\t"+ "Tipo: " + elemento.tipo + "\t" +
                            "Nombre: " + elemento.nombre + "\t" + " Valor: " +elemento.valor);
                }
                JOptionPane.showMessageDialog( parentComponent:null,  message:"Analisis Realizado Exitosamente!",  tit
                //raiz.printArbol(raiz);
            } catch (Exception ex) {
                JOptionPane.showMessageDialog( parentComponent:null,  message:"Error! No se pudo analizar el texto...
                ex.printStackTrace();
                Logger.getLogger( name:FrmPrincipal.class.getName()).log( level:Level.SEVERE,  msg:null,  thrown:ex);
```

Aquí es donde principalmente se tiene la función para ejecutar el programa el cual hará el análisis léxico y sintáctico. En caso de que se haya hecho el análisis correctamente mostrara un mensaje, caso contrario el mensaje será de tipo error.

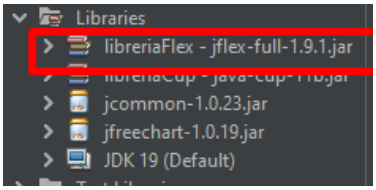
En el archivo “lexerCup.flex” es donde se maneja todo el análisis léxico del programa, se registra cada posible token que pueda surgir, también en caso de errores léxicos se pueden identificar. También se inicializan las listas donde se van a ir agregando tanto los tokens como los errores que mas adelante se pueden utilizar. También se hacen uso de algunas expresiones regulares.

```
L=[a-zA-Z_À-ÿ\u00f1\u00d1]+
D=[0-9]+
NUMERO = {D}+("."{D}+)?
COMLINEA = "!"~(\n|\r)
ESPACIOCADENA = \"([^\"\r\n]*)\"
espacio=[ \t\r\n]+
ESPACIO=[ ]

COMILLA = "\""
IGTILDE = "<!"~"!>"

%{
    public static ArrayList<CError> listaErrores = new ArrayList<>();
    int erroresCount = 1;
    public static ArrayList<CToken> listaTokens = new ArrayList<>();
    int tokensCount = 1;
    CToken token;
%}
%init{

    yyline = 1;
    yycolumn = 1;
%init}
```

```
program {System.out.println("--<Program: " +yytext() + " || linea: "+ yyline + " ||columna: "+ yycolumn+">"); token = new CToken(tokensCount,yytext(),"Program",yyline,yycolumn); listaTokens
end     {System.out.println("--<End: " +yytext() + " || linea: "+ yyline + " ||columna: "+ yycolumn+">"); token = new CToken(tokensCount, yytext(),"End",yyline,yycolumn); listaTokens.add(to
var     {System.out.println("--<Variable: " +yytext() + " || linea: "+ yyline + " ||columna: "+ yycolumn+">"); token = new CToken(tokensCount, yytext(),"Var",yyline,yycolumn); listaTokens.a
double  {System.out.println("--<Var_Tipo: " +yytext() + " || linea: "+ yyline + " ||columna: "+ yycolumn+">"); token = new CToken(tokensCount, yytext(),"Double",yyline,yycolumn); listaToken
char\[\] {System.out.println("--<Var_Tipo: " +yytext() + " || linea: "+ yyline + " ||columna: "+ yycolumn+">"); token = new CToken(tokensCount, yytext(),"Char",yyline,yycolumn); listaTokens
arr     {System.out.println("--<Array: " +yytext() + " || linea: "+ yyline + " ||columna: "+ yycolumn+">"); token = new CToken(tokensCount, yytext(),"Array",yyline,yycolumn); listaTokens.ad
sum     {System.out.println("--<Fun_Arit: " +yytext() + " || linea: "+ yyline + " ||columna: "+ yycolumn+">"); token = new CToken(tokensCount, yytext(),"Fun_Arit",yyline,yycolumn); listaTok
res     {System.out.println("--<Fun_Arit: " +yytext() + " || linea: "+ yyline + " ||columna: "+ yycolumn+">"); token = new CToken(tokensCount, yytext(),"Fun_Arit",yyline,yycolumn); listaTok
mul     {System.out.println("--<Fun_Arit: " +yytext() + " || linea: "+ yyline + " ||columna: "+ yycolumn+">"); token = new CToken(tokensCount, yytext(),"Fun_Arit",yyline,yycolumn); listaTok
div     {System.out.println("--<Fun_Arit: " +yytext() + " || linea: "+ yyline + " ||columna: "+ yycolumn+">"); token = new CToken(tokensCount, yytext(),"Fun_Arit",yyline,yycolumn); listaTok
mod     {System.out.println("--<Fun_Arit: " +yytext() + " || linea: "+ yyline + " ||columna: "+ yycolumn+">");token = new CToken(tokensCount, yytext(),"Fun_Arit",yyline,yycolumn); listaToke
media   {System.out.println("--<Fun_Esta: " +yytext() + " || linea: "+ yyline + " ||columna: "+ yycolumn+">"); token = new CToken(tokensCount, yytext(),"Fun_Esta",yyline,yycolumn); listaTok
mediana {System.out.println("--<Fun_Esta: " +yytext() + " || linea: "+ yyline + " ||columna: "+ yycolumn+">"); token = new CToken(tokensCount, yytext(),"Fun_Esta",yyline,yycolumn); listaTok
moda    {System.out.println("--<Fun_Esta: " +yytext() + " || linea: "+ yyline + " ||columna: "+ yycolumn+">"); token = new CToken(tokensCount, yytext(),"Fun_Esta",yyline,yycolumn); listaTok
varianza {System.out.println("--<Fun_Esta: " +yytext() + " || linea: "+ yyline + " ||columna: "+ yycolumn+">"); token = new CToken(tokensCount, yytext(),"Fun_Esta",yyline,yycolumn); listaTo
min     {System.out.println("--<Fun_Esta: " +yytext() + " || linea: "+ yyline + " ||columna: "+ yycolumn+">"); token = new CToken(tokensCount, yytext(),"Fun_Esta",yyline,yycolumn); listaTok
max     {System.out.println("--<Fun_Esta: " +yytext() + " || linea: "+ yyline + " ||columna: "+ yycolumn+">"); token = new CToken(tokensCount, yytext(),"Fun_Esta",yyline,yycolumn); listaTok
```

De esta forma cuando reconozca un token declarado previamente en este apartado va a mostrar en la consola el token asi como la fila y columna en donde se encuentra, lo agrega también a la lista de tokens declarada previamente y va a retornar el símbolo, la línea y columna del mismo.

Aquí es donde se implementa la librería JFLEX, la cual esta en el apartado de librerías.



Continua con el archivo “Sintax.cup” donde se trabaja el analizador sintáctico. Empieza con la forma en que va a detectar un error y como lo va a imprimir en consola ya que estos son de tipo sintáctico

Primero toma en cuenta los errores y los agrega a la lista ya que esta es para generar los reportes previamente.

```
parser code
{:

    public ArrayList<CError> listaErrores;
    public int erroresCount;

    public void setListaErrores(ArrayList<CError> listaErrores) {
        this.listaErrores = listaErrores;
    }

    public void setErroresCount(int erroresCount) {
        this.erroresCount = erroresCount;
    }

    public void syntax_error(Symbol s){
        System.out.println("Error sintactico: "+s.value+" linea: "+s.left+" columna: "+s.right);
        CError error = new CError(erroresCount, s.value.toString(), "Sintáctico", s.left, s.right);
        listaErrores.add(error);
        erroresCount++;
    }
    public void unrecovered_syntax_error(Symbol s) throws java.lang.Exception{
        System.out.println("Error sintactico unrecovered: "+s.value+" linea: "+s.left+" columna: "+s.right);
    }
```

Después declara los terminales que fueron los declarados en el analizador léxico y los no terminales que son los que se usaran.

```
terminal Linea,Signo_Igual,Signo_Suma,Signo_Resta,Signo_Multiplicacion,Signo_Division,
        Parentesis_Izq, Parentesis_Der,Corchete_Izq, Corchete_Der, Program,End,Variable,Var_Tipo,
        Array,Fun_Arit,Fun_Esta,Fun_Consola,Imprimir,Columna,Ejecutar,Tipo_Grafica,Titulo,Eje_X,
        Eje_Y,Titulo_X,Titulo_Y,Grafica_Label,Grafica_Valores,DosPuntos_Dobles,
        Punto_Coma,Dos_Puntos,Punto,Coma,Signo_Indicador,Signo_Arroba,Identificador,Char_General,COMILLA,Numero,Signo_IndicadorR;



non terminal arbol INICIO_PR, CODIGO, EJECUCION,INSTRUCCION ,D_VARIABLE,D_ARREGLO,D_COMENTARIO,D_GRAFICA ,TIPOEXPR,EXPRE_EST,LISTA_DATOS,
            TIPOARRAY,LISTA_COMENTARIO,START_GRAPH,CONTGRAPH,ATRIBUTOS;
```

Luego se va declarando toda la gramática y al mismo tiempo agregando cada token a un árbol sintáctico.

```
INICIO_PR ::= Program:P1 CODIGO:C End:E Program:P2{:

                arbol inicio = new arbol("INICIO_PR");
                inicio.addHijo(new arbol(P1.toString()));
                inicio.addHijo(C);
                inicio.addHijo(new arbol(E.toString()));
                inicio.addHijo(new arbol(P2.toString()));
                RESULT = inicio;
            :};

CODIGO ::= EJECUCION:E  {:

                arbol codigo = new arbol("CODIGO");
                codigo.addHijo(E);
                RESULT = codigo;
            :}
            |   ;

EJECUCION::= INSTRUCCION:I {:

                arbol ejecucion = new arbol("EJECUCION");
                ejecucion.addHijo(I);
                RESULT = ejecucion;
            :}

            | EJECUCION:E INSTRUCCION:I {:

                arbol ejecucion = new arbol("EJECUCION");
                ejecucion.addHijo(E);
                ejecucion.addHijo(I);
                RESULT = ejecucion;
            :};
```

Aquí para las variables.

```
D_VARIABLE::= Variable:V Dos_Puntos:D Var_Tipo:T DosPuntos_Dobles:DD Identificador:I Signo_Indicador:S TIPOEXPR:EX End:E Punto_Coma:PC {:

        arbol D_variable = new arbol("D_VARIABLE" , 0,0);
        D_variable.addHijo(new arbol(V.toString(), Vleft , Vright));
        D_variable.addHijo(new arbol(D.toString(),Dleft , Dright));
        D_variable.addHijo(new arbol(T.toString(),Tleft , Tright));
        D_variable.addHijo(new arbol(DD.toString(),DDleft , DDright));
        D_variable.addHijo(new arbol(I.toString(),Ileft , Iright));
        D_variable.addHijo(new arbol(S.toString(),Sleft , Sright));
        D_variable.addHijo(EX);
        D_variable.addHijo(new arbol(E.toString(),Eleft , Eright));
        D_variable.addHijo(new arbol(PC.toString(),PCleft , PCright));
        RESULT = D_variable;
:};
```

Aquí para los arrays.

```
D_ARREGLO::= Array:A Dos_Puntos:DP Var_Tipo:V DosPuntos_Dobles:DD Signo_Arroba:SA Identificador:I Signo_Indicador:S EXPRE_EST:LD End:E Punto_Coma:PC {:

        arbol D_arreglo = new arbol("D_ARREGLO" ,0, 0);
        D_arreglo.addHijo(new arbol(A.toString(), Aleft,Aright));
        D_arreglo.addHijo(new arbol(DP.toString(), DPleft,DPright));
        D_arreglo.addHijo(new arbol(V.toString(), Vleft,Vright));
        D_arreglo.addHijo(new arbol(DD.toString(), DDleft,DDright));
        D_arreglo.addHijo(new arbol(SA.toString(), SAleft,SAright));
        D_arreglo.addHijo(new arbol(I.toString(), Ileft,Iright));
        D_arreglo.addHijo(new arbol(S.toString(), Sleft,Sright));
        D_arreglo.addHijo(LD);
        D_arreglo.addHijo(new arbol(E.toString(), Eleft,Eright));
        D_arreglo.addHijo(new arbol(PC.toString(), PCleft,PCright));
        RESULT = D_arreglo;
        :};
```

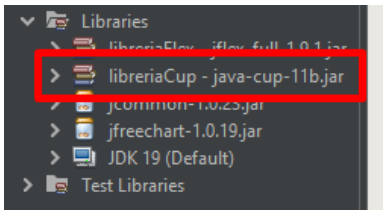
Para la declaración de console.

```
D_COMENTARIO::= Fun_Consola:FC DosPuntos_Dobles LISTA_COMENTARIO:LC {:

        arbol D_comentario = new arbol("D_COMENTARIO");
        D_comentario.addHijo(new arbol(FC.toString()));
        D_comentario.addHijo(LC);
        RESULT = D_comentario;
        :};
```

Y las gráficas.

```
D_GRAFICA::= Tipo_Grafica:TG1 Parentesis_Izq:PI CONTGRAPH:CG Ejecutar:EJ Tipo_Grafica:TG2 End:E1 Punto_Coma:PC1 Parentesis_Der:PD End:E2 Punto_Coma:PC2 {:

            arbol D_grafica = new arbol("D_GRAFICA");
            D_grafica.addHijo(new arbol(TG1.toString()));
            D_grafica.addHijo(new arbol(PI.toString()));
            D_grafica.addHijo(CG);
            D_grafica.addHijo(new arbol(EJ.toString()));
            D_grafica.addHijo(new arbol(TG2.toString()));
            D_grafica.addHijo(new arbol(E1.toString()));
            D_grafica.addHijo(new arbol(PC1.toString()));
            D_grafica.addHijo(new arbol(PD.toString()));
            D_grafica.addHijo(new arbol(E2.toString()));
            D_grafica.addHijo(new arbol(PC2.toString()));
            RESULT = D_grafica;
            :};
```

En este archivo se manejó la librería cup.



Ahora en la clase “arbol.java” es donde manejo mi árbol sintáctico, traigo la información necesaria y la opero ya sea de forma aritmética o estadística, también para declaración de variables y arrays en caso necesite algún valor declarado en otro lado. Los valores van subiendo en mi árbol hasta obtener el resultado.

```
public void run (arbol raiz, ArrayList<CTablaSimb> TablaSim, JTextArea areaConsola) throws IOException{

    for (arbol hijo : raiz.hijos ) {
        run(raiz:hijo,TablaSim,areaConsola);
    }

    if(raiz.etiqueta == "D_VARIABLE"){
        System.out.println("Se encontro : "
        + raiz.hijos.get(index:0).etiqueta
        + " de tipo: " + raiz.hijos.get(index:2).etiqueta
        + " con el valor : " + raiz.hijos.get(index:6).result);

        CTablaSimb simbolo = new CTablaSimb(contadorSimbolos+=1, nombre:raiz.hijos.get(index:4).etiqueta,
         tipo:raiz.hijos.get(index:2).etiqueta,  rol:"Variable", valor:raiz.hijos.get(index:6).result, linea:raiz.hijos.get(index:2).linea, columna:raiz.hijos.get(index:2).columan);
        TablaSim.add(e:simbolo);
    }else if(raiz.etiqueta == "TIPOEXPR" && raiz.hijos.size()==2){
        raiz.result = raiz.hijos.get(index:0).etiqueta + raiz.hijos.get(index:1).etiqueta;
    }else if(raiz.etiqueta == "TIPOEXPR" && raiz.hijos.size()==4){
        raiz.result = String.valueOf(d:this.OpEstadistica(operacion:raiz.hijos.get(index:0).etiqueta.toString().toLowerCase() ,  valores:raiz.hijos.get(index:2).result));
    }else if(raiz.etiqueta == "TIPOEXPR" && raiz.hijos.size()==1){
        if (raiz.hijos.get(index:0).etiqueta.substring(beginIndex:0, endIndex:1).equals(anObject:"\"")){
            raiz.result = raiz.hijos.get(index:0).etiqueta;
        }else{
            try{
                double var = Double.parseDouble(s:raiz.hijos.get(index:0).etiqueta);
                raiz.result = raiz.hijos.get(index:0).etiqueta;
            }catch (Exception e){
                String varAsString = this.getValor(TablaSim,  nombre:raiz.hijos.get(index:0).etiqueta);
                if (varAsString.equals(anObject:"Se produjo un error semantico")){
                    System.out.println(x:"Error!");
                }else{
                    raiz.result = varAsString;
```

En este caso lo voy agregando a mi tabla de símbolos que es un arreglo para más adelante generar los reportes.

En este apartado es donde realizo las operaciones aritméticas.

```
}else if(raiz.etiqueta == "TIPOEXPR" && raiz.hijos.size()==6){
    //System.out.println(raiz.hijos.get(0).etiqueta+" " + raiz.hijos.get(2).result +" " +raiz.hijos.get(4).result);
    if(raiz.hijos.get(index:0).etiqueta.equalsIgnoreCase(anotherString:"sum")){
        raiz.result = String.valueOf(Double.parseDouble(s:raiz.hijos.get(index:2).result) +
                Double.parseDouble(s:raiz.hijos.get(index:4).result));
    }else if(raiz.hijos.get(index:0).etiqueta.equalsIgnoreCase(anotherString:"res")){
        raiz.result = String.valueOf(Double.parseDouble(s:raiz.hijos.get(index:2).result) - Double.parseDouble(s:raiz.hijos.get(index:4).result));
    }else if(raiz.hijos.get(index:0).etiqueta.equalsIgnoreCase(anotherString:"mul")){
        raiz.result = String.valueOf(Double.parseDouble(s:raiz.hijos.get(index:2).result) * Double.parseDouble(s:raiz.hijos.get(index:4).result));
    }else if(raiz.hijos.get(index:0).etiqueta.equalsIgnoreCase(anotherString:"div")){
        raiz.result = String.valueOf(Double.parseDouble(s:raiz.hijos.get(index:2).result) / Double.parseDouble(s:raiz.hijos.get(index:4).result));
    }else if(raiz.hijos.get(index:0).etiqueta.equalsIgnoreCase(anotherString:"mod")){
        raiz.result = String.valueOf(Double.parseDouble(s:raiz.hijos.get(index:2).result) % Double.parseDouble(s:raiz.hijos.get(index:4).result));
    }
```

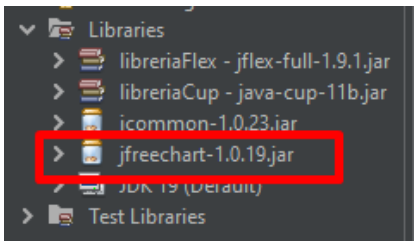
Aquí realizo las operaciones estadísticas.

```java
public static double OpEstadistica(String operacion , String valores){
    String[] omitirComa = valores.split( regex: ",");
    double[] datos = new double[omitirComa.length];

    for(int i = 0; i< omitirComa.length; i++){
        datos[i] = Double.parseDouble(omitirComa[i]);
    }
    System.out.println( x: datos);

    switch (operacion) {
        case "media":
            double getMedia = 0;
            for (double cantidad:datos){
                getMedia += cantidad;
            }
            double resultado = getMedia/datos.length;
            return resultado;
        case "mediana":
            // Ordena los datos
            Arrays.sort( a: datos);
            int n = datos.length;
            if (n % 2 != 0) {
                // Si el número de datos es impar, la mediana es el dato del medio
                return datos[n / 2];
            } else {
                // Si el número de datos es par, la mediana es el promedio de los dos datos de
                return (datos[n / 2 - 1] + datos[n / 2]) / 2.0;
            }
        case "moda":
            // Calcula la moda
            Map<Double, Integer> frecuencia = new HashMap<>();
            for (double dato : datos) {
                frecuencia.put( key: dato, frecuencia.getOrDefault( key: dato,  defaultValue: 0) + 1);
            }
```

Y aquí es donde realizo las graficas ya que puede variar el tipo de grafica solicitado y los datos que se brinden para cada grafica.

```java
public static void tipoGrafica(String tipoGrafica, Map<String, String> contGraph,JTextArea areaConsola) throws IOException {

    switch (tipoGrafica.toLowerCase()) {
        case "graphbar":
            contadorGraficaBar +=1;
           generarGraficaBarra(contGraph, contador: contadorGraficaBar);
            break;
        case "graphline":
            contadorGraficaLine += 1;
            generarGraficaLinea(contGraph, contador: contadorGraficaLine);
            break;
        case "graphpie":
            contadorGraficaPie +=1;
           generarGraficaPie(contGraph, contador: contadorGraficaPie);
            break;
        case "histogram":
            contadorGraficaHisto += 1;
            generarGraficaHistograma(contGraph,areaConsola,  contador: contadorGraficaHisto);
            break;
        default:
            System.out.println( x: "Tipo de grafica indefinido");
    }
}
```

Para las gráficas utilizo un array de tipo Map para agregar los parámetros que utilice cada gráfica y también la librería jfreechart.



Clases:

Utilizo 3 diferentes clases para guardar los datos que se utilizan para generar los reportes.

La primera es “CTablaSimb.java” la cual guarda la información de ciertos tokens que se agregan a la tabla de símbolos.

```java
public class CTablaSimb {
    public int contador;
    public String nombre;
    public String tipo;
    public String rol;
    public String valor;
    int linea = 0;
    int columna = 0;

    public CTablaSimb(int contador,String nombre,String tipo,String rol, String valor, int linea, int columna) {
        this.contador = contador;
        this.nombre = nombre;
        this.tipo = tipo;
        this.rol = rol;
        this.valor = valor;
        this.linea = linea;
        this.columna = columna;
    }
```

La siguiente es “CError.java” la cual guardo la informacion de los errores que se generen ya sea lexicos o sintacticos.

```java
public class CError {
    int contador = 0;
    String error;
    String tipo;
    int linea = 0;
    int columna = 0;

    public CError(int contador, String error, String tipo, int linea, int columna) {
        this.contador = contador;
        this.error = error;
        this.tipo = tipo;
        this.linea = linea;
        this.columna = columna;
    }
```

Y por ultimo esta “CToken.java” la cual guarda la informacion de todos los tokens que vengan.

```java
public class CToken {
    int contador = 0;
    String token;
    String tipo;
    int linea = 0;
    int columna = 0;

    public CToken(int contador,String token, String tipo , int linea, int columna) {
        this.contador = contador;
        this.token = token;
        this.tipo = tipo;
        this.linea = linea;
        this.columna = columna;
    }
```

Ese seria el funcionamiento del programa.