ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

Manual Técnico

PROYECTO 2

LABORATORIO LENGUAJES FORMALES Y DE PROGRAMACIÓN A+

Carlos Daniel Acabal Pérez - 202004724

OCTUBRE 2021

INTRODUCCIÓN

En el presente manual abordará el tema del funcionamiento del código del proyecto 2 el cual consiste en realizar un analizador léxico y sintáctico para reconocer una serie de instrucciones provenientes de un archivo de entrada con extensión .lfp

El manual esta dirigido a técnicos y programadores que manejan el soporte del software y puedan darle mantenimiento o añadir funcionalidades al programa.

INICIO DE LA APLICACIÓN

-Iniciar el programa:
Deberá hacer doble click sobre el archivo "main.py" el cual se encuentra en la carpeta principal

FUNCIONAMIENTO PRINCIPAL

-INTERFAZ:

En la clase "main" se utilizó Tkinter para crear una interfaz con los componentes: Ventana, Etiquetas, Frames, Botones, Text y Menu. Se creó una clase llamada App que contiene toda la interfaz

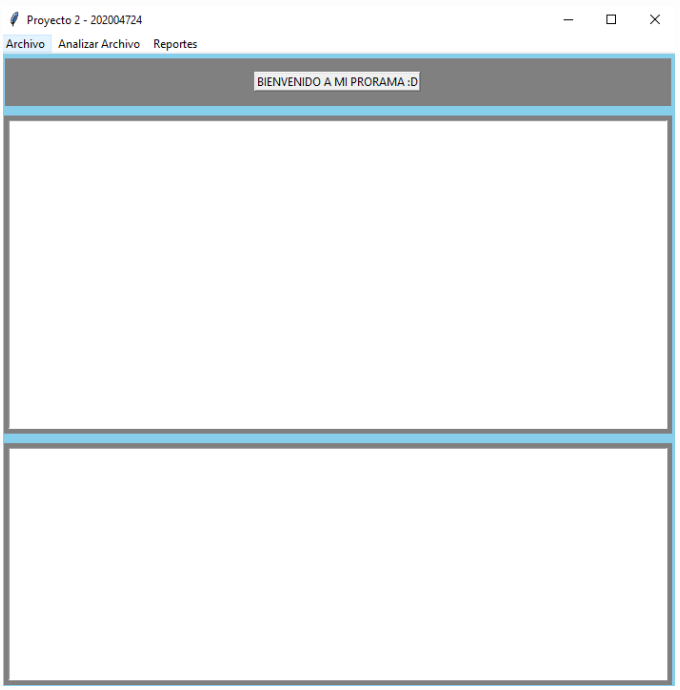
```
#Frames
    #ToolBar
toolBar = tk.Frame(root,width=700,height=50,bg="grey").grid(row=0,padx=5,pady=5)
```

```
#Ventana
root.title("Proyecto 2 - 202004724")
root.config(bg="skyblue")
```

```
#Menu
menubar = Menu(root)
    #Carga y Salir
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="Abrir", command=self.c
```

```
#Botones

buttons = tk.Frame(toolBar,width=700,height=50,bg="grey")
buttons.grid(row=0,padx=5,pady=5)
```



CARGA DE DATOS

Se utilizó Tkinter para obtener la ruta del archivo de entrada a través de un FileDialog.

- Lectura del archivo:

A través del método Open() se obtuvo el contenido como tipo String que se almacenó en la variable entrada

```
def carga(self):
    try:
        path = filedialog.askopenfilename(filetypes=[('Texto plano', '*.lfp')])
        self.entrada = open(path,"r").read()
        self.text.delete("1.0",END)
        self.text.insert(INSERT,self.entrada)
        messagebox.showinfo("Carga","Carga de archivo exitosa")
    except:
        messagebox.showerror("Error","Error al cargar el archivo")
```

CLASES

Se utilizaron varias clases para relacionar objetos
-ANALIZADOR:
Esta clase posee los atributos Lexema, Estado y 2 listas de tokens. Una lista de Tokens y una de error. Posee el método analizar el cual posee el autómata finito determinista programado, éste se encarga de verificar si los símbolos leídos en el archivo de carga pertenece al lenguaje del programa, además llena la lista de tokens y reconocer errores léxicos.
El autómata finito determinista se encuentra en la sección Anexos.
-TOKEN:
Se utilizó un objeto token con los atributos: Tipos, Lexema, Fila y Columna.
-SINTÁCTICO:
Esta clase utiliza la lista de Tokens generada por el Analizador Léxico. Posee el método equals() el cual verifica si el token actual es igual al token esperado para verificar el orden de los tokens. Fue programado en base a una gramática libre de contexto. Utilizando métodos para representar los No Terminales y el método equals() para manejar los Terminales.

La gramática tipo 2 se encuentra en la sección Anexos.

EJECUCIÓN DE INSTRUCCIONES

Los métodos que se encargan de realizar las instrucciones se encuentran en la clase Gestor. Algunos son:

-Almacenamiento de Datos:

Los datos se almacenaron en listas

```
while actual.getTipo()!="Tk_CierraC":
    if actual.getTipo() != "Tk_Coma":
        self.arbol+="Instruccion"+str(n)+"->\""+actual.getTipo()+s
        campos.append(actual.getLexema().replace("\"","").strip())
    else:
        self.arbol+="Instruccion"+str(n)+"->\""+actual.getLexema()
    i+=1
    actual = self.__tokens[i]
```

-Imprimir e Imprimirln:

```
def imprimir(self,cadena):
    self.__consola.insert(INSERT,cadena)
def imprimirln(self,cadena):
    self.__consola.insert(INSERT,cadena+"\n")
```

-Mostrar Datos:

```
def datos(self):
    self.__consola.insert(INSERT,">>>")
    for linea in self.__info[0]:
        self.__consola.insert(INSERT,str(linea)+"\t\t")

    for i in range(1,len(self.__info)):
        self.__consola.insert(INSERT,"\n>>>")
        for j in range(len(self.__info[i])):
            self.__consola.insert(INSERT,self.__info[i][j]+"\t\t")
    self.__consola.insert(INSERT,"\n")
```

Si hay errores Léxicos o Sintácticos, no se ejecutan las instrucciones

GENERAR ARBOL SINTACTICO

El árbol sintáctico se genera durante la ejecución de las instrucciones.
Se utilizó la herramienta Graphviz para crear distintos nodos y relacionarlos entre ellos.
Por cada instrucción se genera un sub árbol, luego se añade a una lista.

```
elif actual.getTipo() == "Tk_conteo":
    self.arbol="->Instruccion"+str(n)+"\n"
    self.conteo()
    self.arbol+="Instruccion"+str(n)+"->"+self.__tokens[i].getTipo()+str(n+i)+"\n"
    self.arbol+="Instruccion"+str(n)+"->"+self.__tokens[i+1].getTipo()+str(n+i)+"\n"
    self.arbol+="Instruccion"+str(n)+"->"+self.__tokens[i+2].getTipo()+str(n+i)+"\n"
    self.arbol+="Instruccion"+str(n)+"->"+self.__tokens[i+3].getTipo()+str(n+i)+"\n"
    self.asintactico.append(self.arbol)
```

Finalmente se recorre la lista para unir los subárboles

```
def reporteArbol(self):
    a = "digraph G {Inicio"
    temp = ""
    for i in range(len(self.asintactico)):
        if i == 0:
            a+="->ListaInstrucciones"+str(i)
            a+=self.asintactico[i]
        else:
            a+="ListaInstrucciones"+str(i-1)
            a+="->ListaInstrucciones"+str(i)
            a+=self.asintactico[i]
    a+="}"
    nam = "ArbolSintactico"
    doc = open(nam+".dot","w")
    doc.write(a)
    doc.close()
```

Se crea un archivo ".dot" para luego convertirlo a una imagen png, la imagen se abre automáticamente

Reportes

- Reporte de Tokens:

Se utilizó la función Open() para escribir un archivo HTML que posea una tabla con los tokens que se identificaron en el archivo de entrada que muestra: el token, lexema, fila y columna del token leído

- Reporte de Errores Léxicos y Sintácticos:

Se escribió en el archivo HTML de reporte de tokens que posee 2 tablas una con los errores léxicos y otra con los errores sintácticos detectados durante el análisis, indicando la fila, columna y lo que provocó el error.

```
def generar(self):
    report = open("Reporte.html","w")
```

```
def printTableToken(self,report,lis):
    report.write("""<div class="container"><table class="table">
                        <thead class="letra">
                            <tr>
                            <th scope="col">Token</th>
                            <th scope="col">Lexema</th>
                            <th scope="col">Fila</th>
                            <th scope="col">Columna</th>
                            </tr>
                        </thead>
                        <tbody class="letra">
                        """)
    for tok in lis:
        if tok.getTipo() != "Final":
            report.write("<tr><td>"+tok.getTipo()+"</td><td>"+str(tok.getLexema())+
    report.write(""" </center></tbody>
                        </table></div>""")
```

Finalmente los reportes se abren automáticamente

```
report.close()
webbrowser.open("Reporte.html")
```

ANEXOS

Símbolos del Lenguaje

No terminal	Símbolos
L	{a, b, ..., z, A, B, ..., Z}
N	{0,1,2,....,9}
S	{ = , ; , { , [,] , } , , , (,)}
R	Cualquier símbolo

Tabla de Tokens

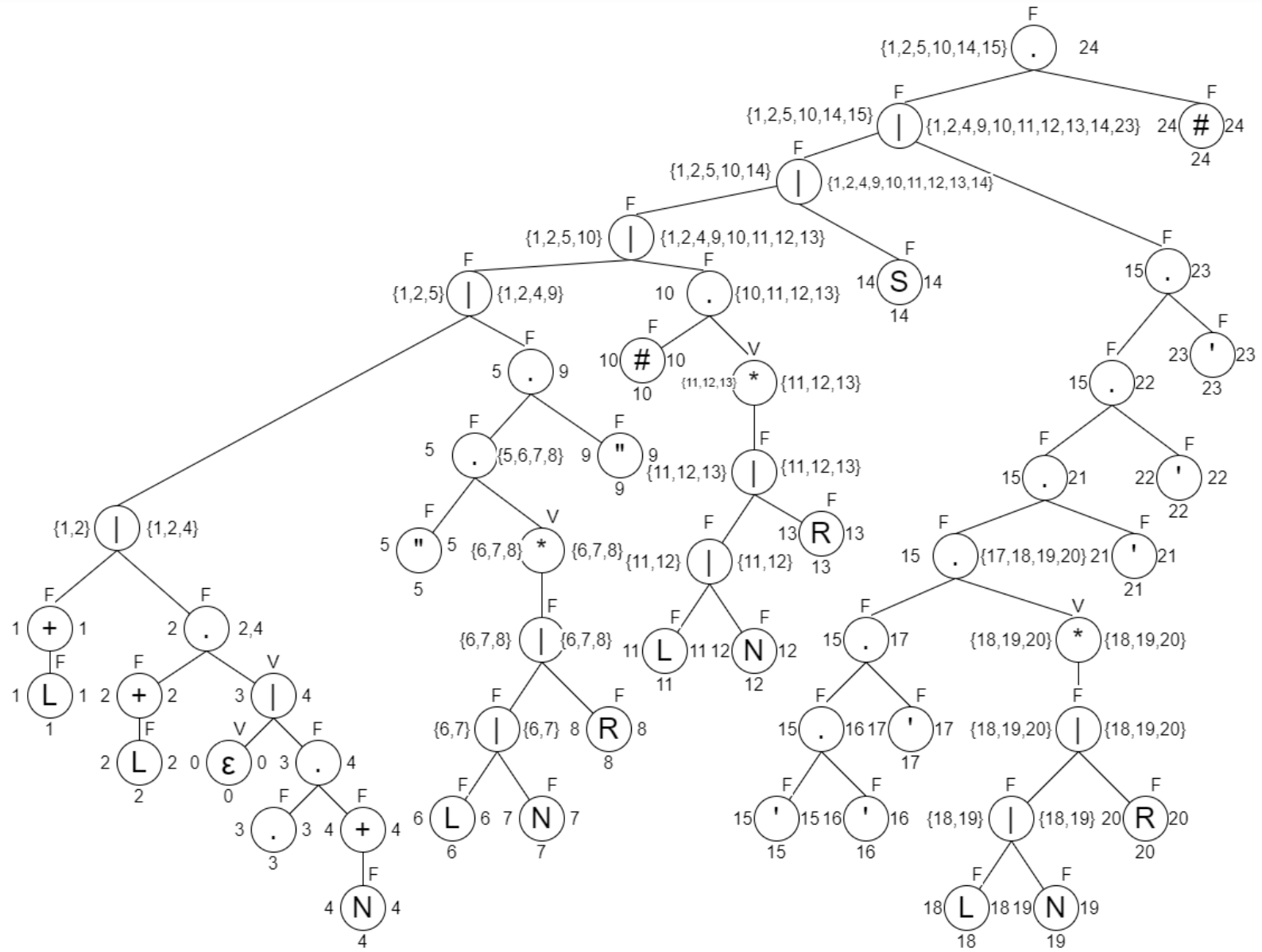
TOKEN	PATRÓN
PALABRARESERVADA	L+
NÚMERO	N+(ε | .N+)
CADENA	" (L | N | R) * "
COMENTARIOLINEA	#(L | N | R) *
COMENTARIOMULTI	''' (L | N | R) * '''
SÍMBOLO	S

Expresión Regular

L+ | N+(ε | .N+) | "(L | N | R) *" | #(L | N | R) * | S | ''' (L | N | R) * '''

ANEXOS

-Método del Arbol:



ANEXOS

- Tabla de siguientes

I	SIMBOLO	SIG(I)
1	L	1,24
2	N	2,3,24
3	.	4
4	N	4,24
5	"	6,7,8,9
6	L	6,7,8,9
7	N	6,7,8,9
8	R	6,7,8,9
9	"	24
10	#	11,12,13,24
11	L	11,12,13,24
12	N	11,12,13,24
13	R	11,12,13,24
14	S	24
15	'	16
16	'	17
17	'	18,19,20,21
18	L	18,19,20,21
19	N	18,19,20,21
20	R	18,19,20,21
21	'	22
22	'	23
23	'	24
24	#	

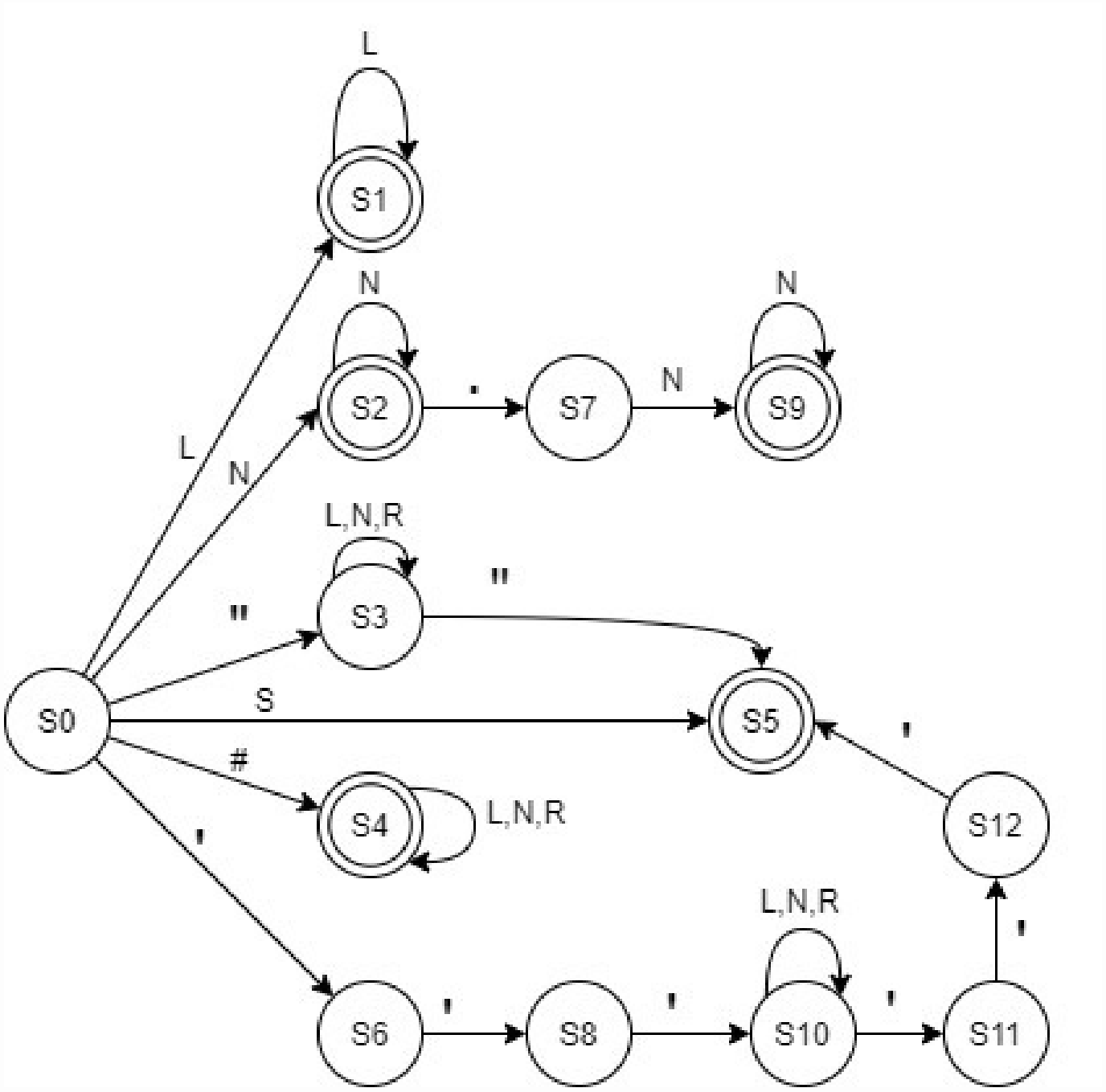
ANEXOS

- Tabla de transiciones

ESTADO\SIMBOLO	L	N	"	#	S	'	.	R
S0={1,2,5,10,14,15}	S1	S2	S3	S4	S5	S6		
S1={1,24}	S1							
S2={2,3,24}		S2					S7	
S3={6,7,8,9}	S3	S3	S5					S3
S4={11,12,13,24}	S4	S4						S4
S5={24}								
S6={16}						S8		
S7={4}		S9						
S8={17}						S10		
S9={4,24}		S9						
S10={18,19,20,21}	S10	S10				S11		S10
S11={22}						S12		
S12={23}						S5		

ANEXOS

- Autómata Finito Determinista



ANEXOS

• Gramática independiente del contexto

- Terminales={ tk_Comentario , tk_ComentarioMultilinea, Tk_AbreP, Tk_Cadena, Tk_CierraP, Tk_PyC, Tk_imprimir, Tk_imprimirln, Tk_promedio, Tk_sumar, Tk_max, Tk_min, Tk_exportarReporte, Tk_datos, Tk_conteo, Tk_contarsi, Tk_coma, Tk_numero, Tk_claves, Tk_igual, Tk_AbreC, Tk_cierraC, Tk_registros, Tk_AbreL, Tk_CierraL }

- No Terminales={<inicio>, <instruccion>, <instruccion2>, <contar>, <claves>, <registros>, <nuevo>, <reservada>, <accion>, <campos>, <otro>, <registro>, <mas>, <Reg>, <otroR> }

Producciones:

<inicio> ::= (tk_Comentario |tk_ComentarioMultilinea|<instruccion>|<instruccion2>|<contar>|<claves>|<registros>)<nuevo>

<nuevo> ::= <inicio>

| epsilon

<intruccion> ::= <reservada> Tk_AbreP Tk_Cadena Tk_CierraP Tk_PyC

<reservada> ::= Tk_imprimir

| Tk_imprimirln

| Tk_promedio

| Tk_sumar

| Tk_max

| Tk_min

| Tk_exportarReporte

<intruccion2> ::= <accion> Tk_AbreP Tk_CierraP Tk_PyC

<accion>::= Tk_datos

| Tk_conteo

<contar> ::= Tk_contarsi Tk_AbreP Tk_Cadena Tk_coma Tk_numeroTk_CierraP Tk_PyC

<claves>::= Tk_claves Tk_igual Tk_AbreC <campos> Tk_cierraC

<campos> ::= Tk_cadena <otro>

<otro> ::= Tk_coma <campo>

| epsilon

<registros>::= Tk_registros Tk_igual Tk_AbreC <registro> Tk_cierraC

<registro> ::= Tk_AbreL<Reg>Tk_CierraL<mas>

<mas>::=<registro>

| epsilon

<Reg>::=Tk_cadena <otroR>

| Tk_numero <otroR>

<otroR> ::= Tk_coma <Reg>

| epsilon