

Analizador Léxico y sintáctico

Álvarez González Ian

García Oviedo Jaasiel Osmar

López López Ulysses

Domínguez Cisneros Alexis Saul

Gpo 1.

Compiladores |
2020-2

Analizador léxico

Análisis del problema:

Se debe elaborar un programa capaz de interpretar las 27 gramáticas dejadas por el profesor, para esto debe crear un analizador Léxico.

Diseño de solución:

Para poder hacer este analizador decidimos usar el lenguaje lex o flex, que, precisamente se basa en el uso de expresiones regulares para la identificación ya sea de patrones o cadenas de lenguaje, lo cual nos ayudará con las cadenas que le queramos introducir. Ahora bien, a continuación, se mostrarán algunos de los diferentes componentes léxicos o tokens que deberá reconocer nuestro analizador léxico, así como sus respectivas expresiones regulares, las cuales hemos visto durante el transcurso del semestre:

Clase o id	Nombre	Expresión regular
0	Reservadas	(dreal car sin struct falso func escribir)
1	digito	[0-9]
2	Reales	[Ee][+-]?[0-9]{1,2}
3	Exponentes	(({enteroNum}\.[0-9]*\.{enteroNum}){exponente}? {enteroNum}{exponente})
4	Letra	[a-zA-z]
4	Letra_	({letra} _)+
5	Id	({letra_} {letra_}{letra_} {letra_}{digito})+
6	Operador	[+ - * / %]
7	Cadena	["]({letra}* {digito}*)+["]
8	Cácter	'.'
9	OpeEsp	[(){}]
10	Condiciona	[&& !]
11	Comentario	[/][*]({letra} {entero})*[n]({letra} {entero})*[*][/]
12	Relacional	[< > <= >= != =]
13	Espacio	[\n\t] +

A continuación, mostraremos las gramáticas que nos dieron:

Gramatica

sin: significa sin tipo, car: tipo carácter

Gramaticas	
1. programa → declaraciones funciones	2. declaraciones → tipo lista_var; declaraciones tipo registro lista_var; declaraciones ε
3. tipo_registro → estructura inicio declaraciones fin	4. tipo → base tipo_arreglo
5. base → ent real dreal car sin	6. tipo_arreglo → [num] tipo arreglo ε
7. lista var → lista var, id id	8. funciones → def tipo id(argumentos) inicio declaraciones sentencias fin funciones ε
9. argumentos → listar_arg sin	10. lista_arg → lista_arg, arg arg
11. arg → tipo_arg id	12. tipo_arg → base param_arr
13. param_arr → [] param_arr ε	14. sentencias → sentencias sentencia sentencia
15. sentencia → si e_bool entonces sentencia fin si e_bool entonces sentencia sino sentencia fin mientras e_bool hacer sentencia fin hacer sentencia mientras e_bool; según (variable) hacer casos predeterminado fin variable := expresion ; escribir expresion ; leer variable ; devolver; devolver expresion; terminar; inicio sentencias fin	16. casos → caso num: sentencia casos caso num: sentencia
17. predeterminado → pred: sentencia ε	18. e_bool → e_bool o e_bool e_bool y e_bool no e_bool (e_bool) relacional verdadero falso
19. relacional → relacional > relacional relacional < relacional relacional <= relacional relacional >= relacional relacional <> relacional relacional = relacional expresion	20. expresion → expresion + expresion expresion - expresion expresion * expresion expresion / expresion expresion % expresion (expresion) variable num cadena caracter
21. variable → id variable comp	22. variable comp → dato est sim arreglo (parametros)
23. dato est_sim → dato est_sim .id ε	24. arreglo → [expresion] arreglo[expresion]
25. parametros → lista_param ε	26. lista_param → lista_param, expresion expresion

Implementación:

```
1  /*
2  * Autores: Dominguez Cisneros Alexis Saul y Garcia Oviedo Jaasiel Osmar
3  * Creado 22/05/2020 by Dominguez Cisneros Alexis Saul
4  * Editado y terminado: 23/05/2020 by Garcia Oviedo Jaasiel Osmar
5  *ultima revision: 24/05/2020 by Dominguez Cisneros Alexis Saul y Garcia Oviedo Jaasiel Osmar
6  *ESTE PROGRAMA LLEVA A CABO LA GENERACION DEL ANALIZADOR LEXICO
7  */
8
9  %{
10     #include <stdio.h>
11     #include <stdlib.h>
12     int num_car=0;
13     int num_linea=0;
14     char linea[20];
15     char cadena[20];
16     FILE *e;
17     int token;
18  %}
19
20  %option noyywrap
21  %option yylineno
22  /*
23  *****EXPRESIONES REGULARES*****
24  */
25  letra [a-zA-Z]
26  digito [0-9]
27  letra_ ({letra}|_)+
28  id ({letra_}|{letra_}{letra_}|{letra_}{digito})+
29  tipo $(ent|real|dreal|car|sin|struct)
30  res #(dreal|car|struct|func|escribir)
31  opesp [(|)|{|}]
32  termina [;]
33  espacio [ \n\t]
34  cadena [""]({letra}*|{digito})*+[""]
35
36  numero ((digito)*.{(digito)+}|((digito))+)
37  coment [/][*]({letra}|{digito})*[\n]({letra}|{digito})*[*][/]
38  opera [+|-|*|/]
39  condi [[|] && | ! ]
40  rela [<|>|<=|>=|!=|=]
41
42  %%
43
44  /*ACCIONES LEXICAS*/
45  /* Se especifican las acciones que se llevaran a cabo cuando se identifiquen cada uno de los tokens acetados por la gramática propuesta.
46  {id} {num_car=num_car+yylen;
47         fputs("\n<id\t,\t",yyout);
48         fputs(yytext,yyout);
49         fputs(">",yyout);
50         yylex();
51         printf("\n<(1)id,%s\n",yytext);
52     }
53  /*Para los casos: res, opesp, opera, condi y rela se retorna una clase lexica por cada uno de los elementos
54  que los conforman y asi estos puedan ser diferenciados e identificados de forma individual por el
55  analizador sintactico.*/
56  {tipo} {num_car=num_car+yylen;
57         fputs("\n<tipo\t,\t",yyout);
58         fputs(yytext,yyout);
59         fputs(">",yyout);
60         yylex();
61         printf("\n<(2)tipo,%s\n",yytext);
62     }
63
64  {res} {num_car=num_car+yylen;
65         fputs("\n<res\t,\t",yyout);
66         fputs(yytext,yyout);
67         fputs(">",yyout);
68         yylex();
```

```

69     printf("\n<(3)res,%s>\n",yytext);
70 }
71
72 ▼ {opesp} {num_car=num_car+yylen;
73     fputs("\n<opesp\t,\t",yyout);
74     fputs(yytext,yyout);
75     fputs(">",yyout);
76     yylex();
77     printf("\n<(4)opesp,%s>\n",yytext);
78 }
79
80 /*los espacios los ignorados*/
81 {espacio} {num_car=num_car+yylen;}
82
83 ▼ {cadena} {num_car=num_car+yylen;
84     fputs("\n<cadena\t,\t",yyout);
85     fputs(yytext,yyout);
86     fputs(">",yyout);
87     yylex();
88     printf("\n<(6)cadena,%s>\n",yytext);
89 }
90
91 ▼ {numero} {num_car=num_car+yylen;
92     fputs("\n<numero\t,\t",yyout);
93     fputs(yytext,yyout);
94     fputs(">",yyout);
95     yylex();
96     printf("\n<(7)numero,%s>\n",yytext);
97 }
98
99 ▼ {opera} {num_car=num_car+yylen;
100     fputs("\n<opera\t,\t",yyout);
101     fputs(yytext,yyout);
102     fputs(">",yyout);

```

```

{coment} {num_char=num_char+yylen;
    fputs("\n<coment\t,\t",yyout);
    fputs(yytext,yyout);
    fputs(">",yyout);
    yylex();
    printf("\n<(9)coment,%s>\n",yytext);
}
{condi} {num_char=num_char+yylen;
    fputs("\n<condicional\t,\t",yyout);
    fputs(yytext,yyout);
    fputs(">",yyout);
    yylex();
    printf("\n<(10)condi,%s>\n",yytext);
}
{rela} {num_char=num_char+yylen;
    fputs("\n<relacional\t,\t",yyout);
    fputs(yytext,yyout);
    fputs(">",yyout);
    yylex();
    printf("\n<(11)rela,%s>\n",yytext);
}
{termina} {num_char=num_char+yylen;
    fputs("\n<termina\t,\t",yyout);
    fputs(yytext,yyout);
    fputs(">",yyout);
    printf("\n<(12)End_Linea,%s>\n",yytext);
}
[ \n\t]+ {}
/*Si se encuentra algun error, son agregados a un archivo en el cual se muestra tanto la linea
como la columna donde se produjo el error lexico.*/
{printf("Error lexico: %s\n", yytext);
    num_char=num_char+yylen;

```

```

103         yylex();
104         printf("\n<(8)opera,%s>\n",yytext);
105     }
106
107     {coment} {num_car=num_car+yylen;
108         fputs("\n<coment\t,\t",yyout);
109         fputs(yytext,yyout);
110         fputs(">",yyout);
111         yylex();
112         printf("\n<(9)coment,%s>\n",yytext);
113     }
114
115     {condi} {num_car=num_car+yylen;
116         fputs("\n<condicional\t,\t",yyout);
117         fputs(yytext,yyout);
118         fputs(">",yyout);
119         yylex();
120         printf("\n<(10)condi,%s>\n",yytext);
121     }
122
123     {rela} {num_car=num_car+yylen;
124         fputs("\n<relacional\t,\t",yyout);
125         fputs(yytext,yyout);
126         fputs(">",yyout);
127         yylex();
128         printf("\n<(11)rela,%s>\n",yytext);
129     }
130
131     {termina} {num_car=num_car+yylen;
132         fputs("\n<termina\t,\t",yyout);
133         fputs(yytext,yyout);
134         fputs(">",yyout);
135         printf("\n<(12)End_Linea,%s>\n",yytext);
136     }

```

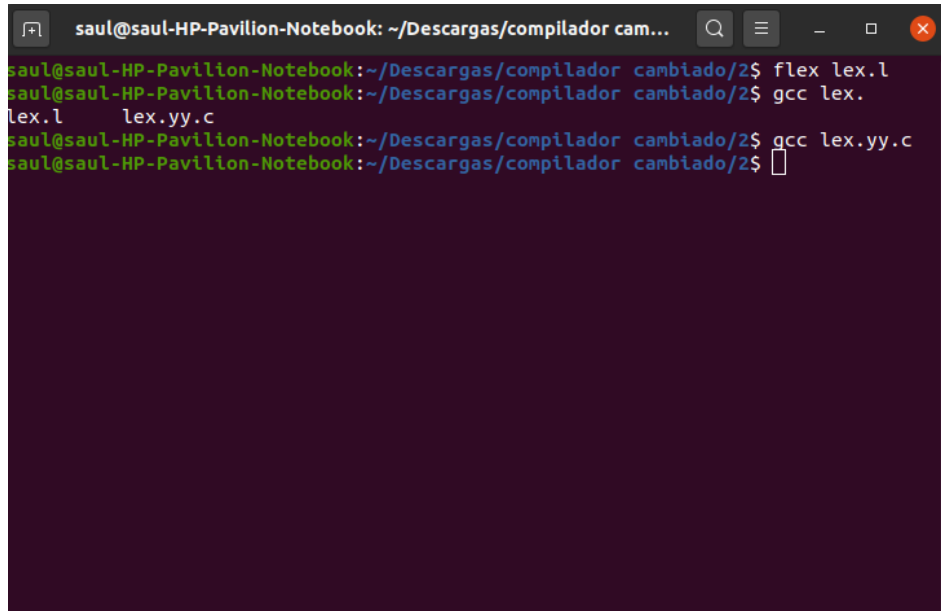
```

139  /*Si se encuentra algun error, son agregados a un archivo en el    cual se muestra tanto la linea
140  como la columna donde se produjo el error lexico.*/
141  {printf("Error lexico: %s\n", yytext);
142      num_car=num_car+yylen;
143      num_linea=yylineno;
144      sprintf(linea, "%i", num_linea);
145      sprintf(cadena, "%i", num_car);
146      fputs("\n<error\t,\t", e);
147      fputs(linea,e);
148      fputs(":", e);
149      fputs(cadena,e);
150  }
151
152  %%
153  /*****Main C*****/
154
155  /*En esta parte se lleva a cabo el procedimiento para abrir/crear y cerrar los archivos utilizados por el analizador
156  lexico. En total se hace uso de dos archivos: Tokens y Errores.*/
157  int main(int argc, char **argv){
158      FILE *f, *t;
159      f=fopen(argv[1], "r");
160      yyin=f;
161      t=fopen("Tokens.txt", "w");
162      e=fopen("Errores.txt", "w");
163      yyout=t;
164      if(yyin==NULL||yyout==NULL){
165          printf("Error\n");
166      }else{
167          yylex();
168      }
169      fclose(yyin);
170      fclose(yyout);
171      fclose(e);
172  }

```

Compilar

1. Para poder ejecutar el programa tenemos que ir a la consola de nuestro equipo.
2. Tenemos que ingresar al fichero donde se encuentra nuestro archivo.
3. Después ingresamos el siguiente comando: *flex "nombre.l"*, si no genera ningún error podemos proceder con el siguiente paso.
4. Ejecutamos el siguiente comando: *gcc lex.yy.c*
5. Se muestra el resultado.

A screenshot of a terminal window with a dark background. The window title is "saul@saul-HP-Pavilion-Notebook: ~/Descargas/compilador cam...". The terminal shows the following commands and output:

```
saul@saul-HP-Pavilion-Notebook:~/Descargas/compilador cambiado/2$ flex lex.l
saul@saul-HP-Pavilion-Notebook:~/Descargas/compilador cambiado/2$ gcc lex.
lex.l      lex.yy.c
saul@saul-HP-Pavilion-Notebook:~/Descargas/compilador cambiado/2$ gcc lex.yy.c
saul@saul-HP-Pavilion-Notebook:~/Descargas/compilador cambiado/2$
```

Compilo correctamente

Código main.c

```
/******Main C******/

/*En esta parte se lleva a cabo el procedimiento para abrir/crear y cerrar los archivos utilizados por el analizador
lexico. En total se hace uso de dos archivos: Tokens y Erroes.*/

int main(int argc, char **argv){
    FILE *f, *t;
    f=fopen(argv[1], "r");
    yyin=f;
    t=fopen("Tokens.txt", "w");
    e=fopen("Errores.txt", "w");
    yyout=t;
    if(yyin==NULL||yyout==NULL){
        printf("Error\n");
    }else{
        yylex();
    }
    fclose(yyin);
    fclose(yyout);
    fclose(e);
}
```

Análisis Sintáctico

Análisis del problema:

El problema principal es poder obtener una correcta interpretación de una serie de gramáticas propuestas por el profesor, esto forma parte de un conjunto de problemas, los cuales tienen como objetivo un proyecto final el cual es construir un compilador.

En el análisis sintáctico se deben tomar varias decisiones, la gramática en algunas partes presenta ambigüedad, sin embargo; esta se puede ignorar utilizando en bison la precedencia de operadores y dejar que el programa la resuelva de manera automática o eliminar la ambigüedad antes de transcribir la gramática en bison.

Otra consideración que se debe tener en cuenta para resolver la ambigüedad del if - else, es que se debe asignar una precedencia a else como si fuera el operador de mayor precedencia. Ahora bien, teniendo en cuenta las gramáticas que se nos dieron, hicimos sus producciones, así como sus respectivas reglas semánticas como se mostrarán a continuación:

Regla de Producción	Regla Semántica
1) $P \rightarrow D_F$	STS.push(nuevaTS()) STT.push(nuevaTT()) dir = 0 P.codigo = S.codigo TOS = nuevaTOS()
2) $D \rightarrow T L_V$	Tipo = T.tipo
2) $D \rightarrow \epsilon$	
3) $T_R \rightarrow \text{struct } \{D\}$	STS.push(nuevaTS()) STT.push(nuevaTT()) S.dir.push(dir) dir=0 dir = S.dir.pop() TS = STS.pop() TT = STT.pop() TS.TT= TT T.tipo = STT.getCima().append('struct', tam, TS)
4) $T \rightarrow B T_A$	B = B.base T.tipo = T_A.tipo
5) $B \rightarrow \text{ent}$	B.tipo = ent
5) $B \rightarrow \text{real}$	B.tipo = real
5) $B \rightarrow \text{dreal}$	B.tipo = dreal
5) $B \rightarrow \text{car}$	B.tipo = car
5) $B \rightarrow \text{sin}$	B.tipo = sin
6) $T_A \rightarrow [\text{num}] T_A1$	Si num.tipo = ent Entonces

	Si num.dir > 0 Entonces T_A.tipo = TT.append("array", num.dir, T_A1.tipo) Sino Error("El indice debe ser mayor a 0") Fin Si Sino Error("El índice debe de ser entero") Fin Si
6) $T_A \rightarrow \epsilon$	T_A.tipo = base
7) $L_V \rightarrow L_V1, id$	Si !TS.existe(id) Entonces STS.getCima().append(id, dir, T, 'var', nulo, -1) dir ← dir + STT.getCima().getTam(Tipo) Sino Error("El id ya existe") Fin Si
7) $L \rightarrow id$	Si !TS.existe(id) Entonces STS.getCima().append(id, dir, Tipo, 'var', nulo, -1) dir ← dir + STT.getCima().getTam(Tipo) Sino Error("Ya fue declarado el id") Fin Si
8) $F \rightarrow \text{def } T \text{ id (ARG) \{DS\} } F$	Si !STS.getCima().existe(id) Entonces STS.push(nuevaTS()) S.dir.push(dir) dir=0 L_R = nuevaLista() Si cmpRet(L_R, T.tipo) Entonces L =nuevaEtiqueta() backpatch(S.nextlist, L) F.codigo = etiqueta(id) S.codigo etiqueta(L) Sino Error("El valor no corresponde al tipo de la función") Fin Si STS.pop() dir = S_D.pop() Sino Error("El id ya fue declarado") Fin Si
8) $F \rightarrow \epsilon$	
9) $ARG \rightarrow L_ARG$	ARG.lista = L_ARG.lista ARG.num = L_ARG.num
9) $ARG \rightarrow \epsilon$	ARG.lista = nulo ARG.num = 0
10) $L_ARG \rightarrow L_ARG1, T \text{ id } ARG$	L_ARG.lista = L_ARG1.lista L_ARG.lista.append(T.tipo)

	L_ARG.num = L_ARG1.num +1
10) L_ARG → T id ARG	L_ARG.lista = nuevaLista() L_ARG.lista.append(T.tipo) L_ARG.num = L_ARG1.num +1
11) ARG → T_ARG id	Si STS.getCima().getId(id)= -1 Entonces STS.getCima().addSym(id,tipo,dir,"var") dir = dir + STT.getCima().getTam(T) Sino Error("El identificador ya fue declarado") Fin ARG.tipo = T_ARG.tipo
12) T_ARG → B P_A	B = B.tipo T_ARG.tipo = P_A.tipo
13) P_A → [] P_A1	P_A.tipo = STT.append("array",- ,P_A1.tipo, null)
13) P_A → ε	P_A.tipo = B
14) S → S1S2	L =nuevaEtiqueta() backpatch(S1.nextlist, L) S.nextlist = S2.nextlist S.codigo = S1.codigo etiqueta(L) S2.codigo
14) S → S1	
15) S → si (E_B) entonces S1 fin	L =nuevaEtiqueta() backpatch(E_B.truelist, L) S.nextlist= combinar(E_B.falselist, S1.nextlist) S.codigo = E_B.codigo etiqueta(L) S1.codigo
15) S → si (E_B) entonces S1 sino S2 fin	L1 = nuevaEtiqueta() L2 = nuevaEtiqueta() backpatch(E_B.truelist, L1) backpatch(E_B.falselist, L2) S.nextlist = combinar(S1.nextlist, S2.nextlist) S.codigo = E_B.codigo etiqueta(L1) S1.codigo gen('goto' S1.nextlist[0]) etiqueta(L2) S2.codigo
15) S → mientras (E_B) hacer S1 fin	L1 = nuevaEtiqueta() L2 = nuevaEtiqueta() backpatch(S1.nextlist, L1) backpatch(B.truelist, L2) S.nextlist = B.falselist S.codigo = etiqueta(L1) B.codigo etiqueta(L2) S1.codigo gen('goto' S1.nextlist[0]) S2.codigo
15) S →segun (V) hacer CP fin	
15) S → V := E;	S.codigo = E.codigo V '=' E.dir
15) S → escribir E;	S.codigo = gen("print") E.codigo
15) S → leer V;	S.codigo=gen("scan" E.dir) S.listnext = nulo
15) S →devolver E;	S.nextlist = nulo

	L_R.append(E.tipo) S.codigo = gen(return E.dir)
15) S → devolver;	S.nextlist = nulo S.codigo = gen(return)
15) S → terminar;	L = nuevaEtiqueta() S.codigo=gen('goto' L) S.nextlist = nuevaLista() S.nextlist.add(L)
16) CASS → caso num: S CASS1	
16) CASS → caso num: S	
17) PRED → PRED: S	
17) PRED → ε	
18) E_B → E_B1 E_B2	L = nuevaEtiqueta() backpatch(E_B1.falselist, L) E_B.truelist = combinar(E_B1.truelist, E_B2.truelist) E_B.falselist = E_B2.falselist E_B.codigo = E_B1.codigo etiqueta(L) E_B2.codigo
18) E_B → E_B1 && E_B2	L = nuevaEtiqueta() backpatch(E_B1.truelist, L) E_B.truelist = E_B2.truelist E_B.falselist = combinar(E_B1.falselist, E_B2.falselist) E_B.codigo = E_B1.codigo etiqueta(L) E_B2.codigo
18) E_B → ! E_B1	B.truelist =B1.falselist B.falselist = B1.truelist B.codigo = B1.codigo
18) E_B → E1 R E2	t0 = nuevoIndice() t1 = nuevoIndice() B.truelist=crearLista(t0) B.falselist=crearLista(t1) B.codigo = gen('if' E1.dir R.op E2.dir 'goto' t0) gen('goto' t1)
18) E_B →verdadero	t0 = nuevoIndice() E_B.truelist = nuevaLista(t0) E_B.codigo = gen('goto' t0)
18) C → falso	t0 = nuevoIndice() E_B.falselist = crearLista(t0) E_B.codigo = gen('goto' t0)
19) R → R1 < R2	R.dir = nuevaTemp R.tipo = maximo(R1.tipo , R2.tipo) t1= ampliar(R1.dir,R1.tipo,R.tipo) t2= ampliar(R2.dir, R2.tipo,R.tipo) R.codigo = gen(R.dir=' t1'<'t2)
19) R → R1 > R2	R.dir = nuevaTemp R.tipo = maximo(R1.tipo , R2.tipo)

	t1= ampliar(R1.dir,R1.tipo,R.tipo) t2= ampliar(R2.dir, R2.tipo,R.tipo) R.codigo = gen(R.dir=' t1'>'t2)
19) $R \rightarrow R1 \geq R2$	R.dir = nuevaTemp R.tipo = maximo(R1.tipo , R2.tipo) t1= ampliar(R1.dir,R1.tipo,R.tipo) t2= ampliar(R2.dir, R2.tipo,R.tipo) R.codigo = gen(R.dir=' t1'>='t2)
19) $R \rightarrow R1 \leq R2$	R.dir = nuevaTemp R.tipo = maximo(R1.tipo , R2.tipo) t1= ampliar(R1.dir,R1.tipo,R.tipo) t2= ampliar(R2.dir, R2.tipo,R.tipo) R.codigo = gen(R.dir=' t1'<='t2)
19) $R \rightarrow R1 \Leftrightarrow R2$	R.dir = nuevaTemp R.tipo = maximo(R1.tipo , R2.tipo) t1= ampliar(R1.dir,R1.tipo,R.tipo) t2= ampliar(R2.dir, R2.tipo,R.tipo) R.codigo = gen(R.dir=' t1'<>'t2)
19) $R \rightarrow R1 = R2$	R.dir = nuevaTemp R.tipo = maximo(R1.tipo , R2.tipo) t1= ampliar(R1.dir,R1.tipo,R.tipo) t2= ampliar(R2.dir, R2.tipo,R.tipo) R.codigo = gen(R.dir=' t1'='t2)
19) $R \rightarrow E$	R.dir =R.dir R.codigo =E.codigo
20) $E \rightarrow E1 + E2$	E.tipo = maximo(E1.tipo, E2.tipo) E.dir = nuevaTemp() t1 = ampliar(E1.dir, E1.tipo, E.tipo) t2 = ampliar(E2.dir, E2.tipo, T.tipo) E.codigo = E2.codigo T.dir '=' t1 '+' t2
20) $E \rightarrow E1 - E2$	E.tipo = maximo(E1.tipo, E2.tipo) E.dir = nuevaTemp() t1 = ampliar(E1.dir, E1.tipo, E.tipo) t2 = ampliar(E2.dir, E2.tipo, T.tipo) E.codigo = E2.codigo T.dir '=' t1 '-' t2
20) $E \rightarrow E1 * E2$	E.tipo = maximo(E1.tipo, E2.tipo) E.dir = nuevaTemp() t1 = ampliar(E1.dir, E1.tipo, E.tipo) t2 = ampliar(E2.dir, E2.tipo, T.tipo) E.codigo = E2.codigo T.dir '=' t1 '*' t2
20) $E \rightarrow E1 / E2$	E.tipo = maximo(E1.tipo, E2.tipo) E.dir = nuevaTemp() t1 = ampliar(E1.dir, E1.tipo, E.tipo) t2 = ampliar(E2.dir, E2.tipo, T.tipo) E.codigo = E2.codigo T.dir '=' t1 '/' t2
20) $E \rightarrow E1 \% E2$	E.tipo = maximo(E1.tipo, E2.tipo)

	E.dir = nuevaTemp() t1 = ampliar(E1.dir, E1.tipo, E.tipo) t2 = ampliar(E2.dir, E2.tipo, T.tipo) E.codigo = E2.codigo T.dir '=' alfa1 ' %' alfa2
20) $E \rightarrow (E1)$	
20) $E \rightarrow V$	Si TS.existe(variable) Entonces E.dir =V.dir E.tipo = TS.getTipo(V) Sino error("La variable no ha sido declarada") Fin Si
20) $E \rightarrow \text{cadena}$	E.tipo = cadena E.dir =TOS.add(cadena)
20) $E \rightarrow \text{num}$	E.tipo = num.tipo E.dir = num.val
20) $E \rightarrow \text{car}$	E.tipo = car E.dir =TOS.add(car)
21) $V \rightarrow \text{id } V_C$	
22) $V_C \rightarrow D_S_T$	
22) $V_C \rightarrow A$	V_C.dir =A.dir V_C.base = A.base V_C.tipo =A.tipo
22) $V_C \rightarrow (P)$	V_C.lista =P.lista V.num = P.num
23) $D_S_T \rightarrow D_S_T.\text{id}$	
23) $D_S_T \rightarrow \epsilon$	
24) $A \rightarrow \text{id } [E]$	A.dir = nuevaTemp() A.base = id A.tipo = TT.getTipoBase(id.tipo) A.codigo = E.codigo A.dir '=' E.dir 'x' TT.getTam(A.tipo)
24) $A \rightarrow A1 [E]$	A.base = A1.base A.tipo = TT.getTipoBase(A1.tipo) Temp = nuevaTemp() A.dir = nuevaTemp() A.codigo = A1.codigo E.codigo temp '=' E.dir 'x' TT.getTam(A.tipo) A.dir '=' A1.dir '+' temp
25) $P \rightarrow L_P$	P.lista =L_P.lista P.num =L_P.num
25) $P \rightarrow \epsilon$	
26) $L_P \rightarrow L_P1, E$	L_P.lista = L_P1.lista L_P.lista.append(E.tipo) L_P.num = L_P1.num +1

1	$P \rightarrow DF$
2	$D \rightarrow T_L_V \mid \varepsilon$
3	$B \rightarrow \text{ent} \mid \text{real} \mid \text{dreal} \mid \text{car} \mid \text{sin} \mid \text{struct } \{D\}$
4	$L_V \rightarrow L_V1, \text{id} \mid \text{id}$
5	$T_A \rightarrow [\text{num}] T_A1 \mid \varepsilon$
6	$F \rightarrow \text{def } T \text{ id } (ARG) \{DS\} F \mid \varepsilon$
7	$ARG \rightarrow L_ARG \mid \varepsilon$
8	$L_ARG \rightarrow L_ARG1, T \text{ id_ARG} \mid T_ARG \text{ id}$
9	$P_A \rightarrow [] P_A1 \mid \varepsilon$
10	$S \rightarrow S1S2 \mid \text{si } (E_B) \text{ entonces } S1 \text{ fin} \mid \text{si } (E_B) \text{ entonces } S1 \text{ sino } S2 \text{ fin} \mid$ $\text{mientras } (E_B) \text{ hacer } S1 \text{ fin} \mid V := E; \mid \text{devolver } E; \mid \text{devolver}; \mid \{S\} \mid \text{segun } (V)$ $\text{hacer CP fin} \mid \text{terminar}; \mid \text{escribir } E;$
11	$CASS \rightarrow \text{caso num: } S \text{ CASS1}$
12	$PRED \rightarrow PRED: S \mid \varepsilon$
13	$A \rightarrow \text{id } [E] \mid A1 [E]$
14	$E \rightarrow E1 + E2 \mid E1 - E2 \mid E1 * E2 \mid E1 / E2 \mid E1 \% E2 \mid V \mid \text{cadena} \mid \text{num} \mid \text{car}$
15	$P \rightarrow \varepsilon \mid L_P$
16	$L_P \rightarrow L_P1, E$
17	$E_B \rightarrow E_B1 \parallel E_B2 \mid E_B1 \&\& E_B2 \mid ! E_B1 \mid E1 R E2 \mid \text{verdadero} \mid \text{falso}$
18	$R \rightarrow R1 < R2 \mid R1 > R2 \mid R1 \geq R2 \mid R1 \leq R2 \mid R1 != R2 \mid R1 = R2$

Ilustración 2: Árbol

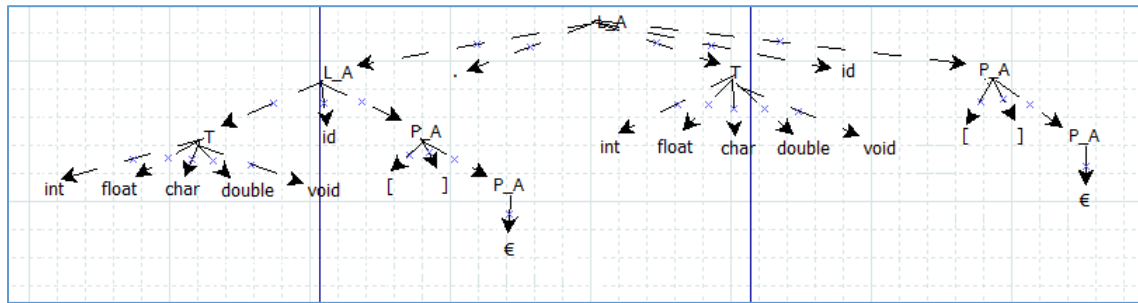


Ilustración 3:Árbol 3

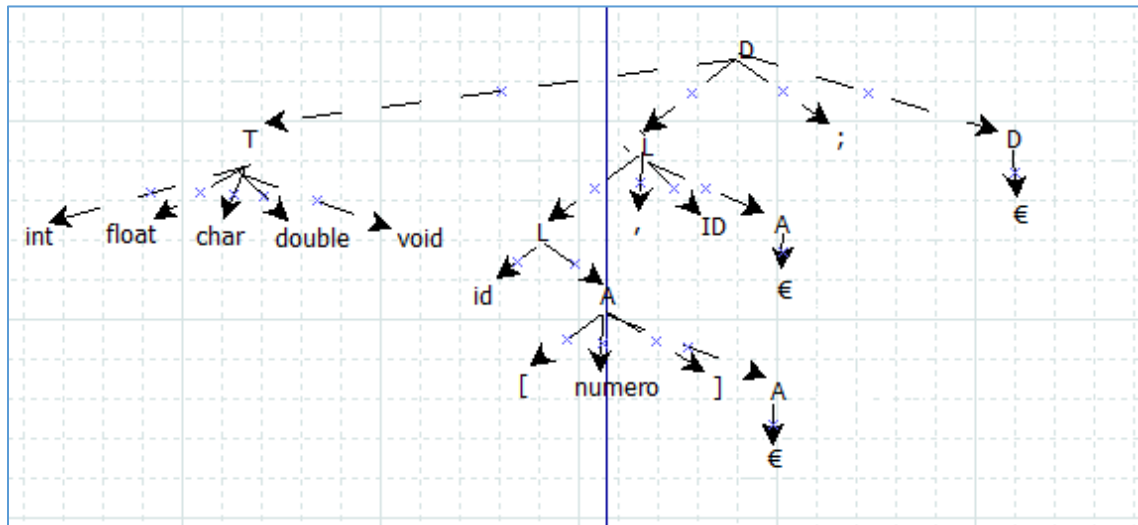


Ilustración 4: Árbol 4

Implementación:

```

1  /*
2      *File: parser.y
3      *Autores: Álvarez González Ian Arturo y López López Ulysses
4      *Creado: 22/05/2020
5      *Editado y terminado: 23/05/2020
6      *Última revisión: 24/05/2020
7      *Este programa lleva a cabo la generación del analizador sintáctico
8  */
9  %{
10     #include <stdio.h>
11     extern int yylex();
12     void yyerror (char *)
13  %}
14
15  /-----DECLARACION DE TOKENS-----*/
16  /* PYC : Punto Y Coma
17     MAS : +
18     MEN : -
19     POR : *
20     DIV : /
21     MOD : %
22     MENQUE : <
23     MAYQUE : >
24     MENIGUAL : <=
25     MAYIGUAL : >=
26     DIF : !=
27     IGUAL : ==
28     NEG : !
29     PARA : (
30     PARC : )
31     CORA : [
32     CORC : ]
33     LLA : {
34     LLC : }
35     YY : &&

```

```

36      OO : ||
37      COMA : ,
38      DP : :
39      PUNTO : .*/
40
41      %token ID
42      %token ENT REAL DREAL CAR SIN STRUCT
43      %token FALSO FUN ESCRIBIR VERDADERO SI SINO MIENTRAS SEGUN DEVOLVER TERMINAR CASO PREDETERMINADO
44      %nonassoc PARA PARC LLAA LLAC CORA CORC
45      %token CADENA
46      %token CARACTER
47      %token NUMERO
48      %right NEG
49      %left POR DIV MOD
50      %left MAS MEN
51      %token MENQUE MAYQUE MENIGUAL MAYIGUAL
52      %token IGUAL DIF
53      %left YY
54      %left OO
55      %right ASIGNA
56      %token COMENT
57      %token PYC
58      %token COMA DP PUNTO
59
60
61      %start prog
62
63      %%
64      /*-----DECLARACION DE LA GRAMATICA-----*/
65      /* A = Arreglo
66         ARG = Argumento
67         L_A = Lista de Argumentos
68         P_A = Parte Arreglo
69         P_I = Parte Izquierda
70         CAS = Casos
71         PRED= predeterminado
72         V_A = var_arreglo
73         P = Parámetros
74         L_P = lista_parametros
75         prog = programa
76         decl = declaraciones
77         array = arreglo
78         arg = argumentos
79         pred = predeterminado
80      */
81
82      /* P → DF */
83      prog : decl funcion;
84
85      /* D → T L; | E */
86      decl : base lista PYC | ;
87
88      /* B → ent | real | dreal | car | sin | struct {D} */
89      base : ENT | REAL | DREAL | CAR | SIN | STRUCT LLAA decl LLAC;
90
91      /* L → L, id A | id A */
92      lista : lista COMA ID array | ID array;
93
94      /* A → [ num ] A | E */
95      array : CORA NUMERO CORC array | ;
96
97      /* F → func T id ( ARG ) { D } F | E */
98      funcion : FUN base ID PARA arg PARC LLAA decl sent LLAC funcion | ;
99
100     /* ARG → L_A | E */
101     arg : lista_arg | ;
102
103     /* L_A → L_A , T id P_A | T id P_A */
104     lista_arg : lista_arg COMA base ID parte_array | base ID parte_array;

```



```

106 /* P_A → CORA CORC P_A | ε */
107 parte_array : CORA CORC parte_array | ;
108
109 /* S → SS | si ( E_B ) entonces S | si ( C ) S sino S | mientras ( C ) que S | devolver E; | devolver; | { S } |
110 MIENTRAS ( E ) { CASO PRED } | terminar; | escribir E; */
111 sent : sent sent | SI PARA ebool PARC sent | SI PARA ebool PARC sent SINO sent | MIENTRAS PARA ebool PARC sent |
112 parte_izq ASIGNA exp PYC | DEVOLVER exp PYC | DEVOLVER PYC | LLAA sent LLAC | MIENTRAS PARA exp PARC LLAA caso pred LLAC |
113 TERMINAR PYC | ESCRIBIR exp PYC;
114
115 /* CAS → case: num S PRED | ε */
116 caso : CASO DP NUMERO sent pred | ;
117
118 /* PRED → default: S | ε */
119 pred : PREDETERMINADO DP sent | ;
120
121 /* P_I → id | V_A | id.id */
122 parte_izq : ID | var_arg | ID PUNTO ID;
123
124 /* V_A → id [ E ] | V_A [ E ] */
125 var_arg : ID CORA exp CORC | var_arg CORA exp CORC;
126
127 /* E → E + E | E - E | E * E | E / E | E % E | V_A | cadena | num | caracter | id (PAR) */
128 exp : exp MAS exp | exp MEN exp | exp POR exp | exp DIV exp | exp MOD exp | var_arg | CADENA | NUMERO | CARACTER | ID PARA param PARC;
129
130 /* PAR → E | L_P */
131 param : ; | lista_param;
132
133 /* L_P = L_P , E | ε */
134 lista_param : lista_param COMA exp | exp;
135
136 /* E_B → E_B || E_B | E_B && E_B | ! E_B | ( E_B ) | E R E | true | false */
137 ebool : ebool OO ebool | ebool YY ebool | NEG ebool | PARA ebool PARC | exp rel exp | VERDADERO | FALSO;
138
139 /* R → R < R | R > R | R >= R | R <= R | R != R | R == R */
140 rel : MENQUE | MAYQUE | MAYIGUAL | MENIGUAL | DIF | IGUAL ;

```

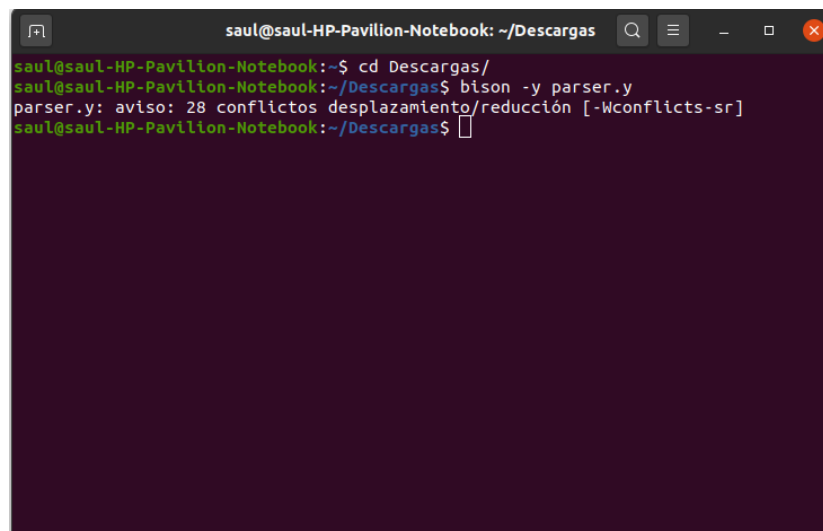
```

141
142 %%
143
144 /*DEFINICION FUNCION ERROR*/
145 /*La funcion error lo que va a hacer es una llamada de función en un archivo c que implementa la función error*/
146
147 void yyerror (s) char *s {
148     printf ("Error: %s\n",s);
149 }
150
151

```

Compilación:

1. Para poder ejecutar el programa tenemos que ir a la consola de nuestro equipo.
2. Tenemos que ingresar al fichero donde se encuentra nuestro archivo.
3. Después ingresamos el siguiente comando: *bison -y "nombre.y"*.



```

saul@saul-HP-Pavillon-Notebook: ~/Descargas
saul@saul-HP-Pavillon-Notebook:~$ cd Descargas/
saul@saul-HP-Pavillon-Notebook:~/Descargas$ bison -y parser.y
parser.y: aviso: 28 conflictos desplazamiento/reducción [-Wconflicts-sr]
saul@saul-HP-Pavillon-Notebook:~/Descargas$

```