

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	Cáacter	'.'
9	OpeEsp	[(){}]
10	Condicional	[&& !]
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:

19

```
1  %{
2
3      #include <stdio.h>
4      #include <string.h>
5      #include <stdlib.h>
6      #include "yacc_p.tab.h"
7
8  %}
9
10 %option noyywrap
11 %option yylineno
12
13
14 dígito [0-9]+
15 ent_def [+]?{dígito}
16 exp [Ee][+-]?{dígito}+
17 real_def ({dígito}.?{dígito}{exp}|{dígito}?.{dígito}{exp}|{dígito}.{dígito}?{exp}?)[Ff]
18 dreal_def ({dígito}.?{dígito}{exp}|{dígito}?.{dígito}{exp}|{dígito}.{dígito}?{exp}?)[Dd]
19 letra [a-zA-Z]
20 id (_|{letra})(_|{letra}|{dígito})){0,31}
21
22 car_def '([^\'])*'
23 cadena \"([^\"])*\"
24
25 registro [Rr][Ee][Gg][Ii][Ss][Tt][Rr][Oo]
26 inicio [Ii][Nn][Ii][Cc][Ii][Oo]
27 fin [Ff][Ii][Nn]
28 ent [Ee][Nn][Tt]
29 real [Rr][Ee][Aa][Ll]
30 dreal [Dd][Rr][Ee][Aa][Ll]
31 car [Cc][Aa][Rr]
32 sin [Ss][Ii][Nn]
33 def [Dd][Ee][Ff]
34 si [Ss][Ii]
35 entonces [Ee][Nn][Tt][Oo][Nn][Cc][Ee][Ss]
36 sino [Ss][Ii][Nn][Oo]
37 mientras [Mm][Ii][Ee][Nn][Tt][Rr][Aa][Ss]
38 hacer [Hh][Aa][Cc][Ee][Rr]
39 escribir [Ee][Ss][Cc][Rr][Ii][Bb][Ii][Rr]
40 leer [Ll][Ee][Ee][Rr]
41 devolver [Dd][Ee][Vv][Oo][Ll][Vv][Ee][Rr]
42 terminar [Tt][Ee][Rr][Mm][Ii][Nn][Aa][Rr]
43 verdadero [Vv][Ee][Rr][Dd][Aa][Dd][Ee][Rr][Oo]
44 falso [Ff][Aa][Ll][Ss][Oo]
45
46 %x com1
47 %x com2
48
49
50 %%
51
52
53
54
55 {ent_def} {yyval.line = yylineno;yyval.num.tipo = 1;yyval.num.ival = atoi(yytext);return NUM;}
56 {real_def} {yyval.line = yylineno;yyval.num.tipo = 2;yyval.num.fval = atof(yytext);return NUM;}
57 {dreal_def} {yyval.line = yylineno;yyval.num.tipo = 3;yyval.num.dval = atof(yytext);return NUM;}
58 {car_def} {yyval.line = yylineno;yyval.car.tipo = 4;sprintf(yyval.sval,"%c",yytext[1]);return CHARACTER;}
59 {ent} {yyval.line = yylineno;return ENT;}
60 {dreal} {yyval.line = yylineno;return DREAL;}
61 {real} {yyval.line = yylineno;return REAL;}
62 {car} {yyval.line = yylineno;return CAR;}
63 {sin} {yyval.line = yylineno;return SIN;}
64 {registro} {yyval.line = yylineno;return REGISTRO;}
65 {inicio} {yyval.line = yylineno;return INICIO;}
66 {def} {yyval.line = yylineno;return DEF;}
67 {sino} {yyval.line = yylineno;return SINO;}
68 {si} {yyval.line = yylineno;return SI;}
69 {devolver} {yyval.line = yylineno;return DEVOLVER;}
70 {fin} {yyval.line = yylineno;return FIN;}
71 {entonces} {yyval.line = yylineno;return ENTONCES;}
72 {verdadero} {yyval.line = yylineno;return VERDADERO;}
73 {falso} {yyval.line = yylineno;return FALSO;}
74 {mientras} {yyval.line = yylineno;return MIENTRAS;}
75 {hacer} {yyval.line = yylineno;return HACER;}
```

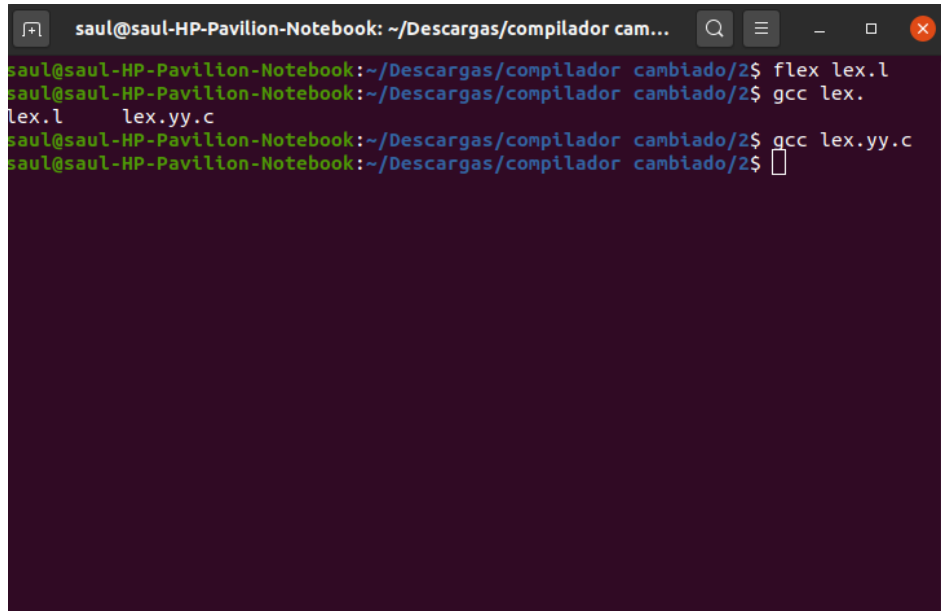
```

96
97 {leer} {yylval.line = yylineno;return LEER;}
98
99 {escribir} {yylval.line = yylineno;return ESCRIBIR;}
100
101 "o" {yylval.line = yylineno;return O;}
102
103 "y" {yylval.line = yylineno;return Y;}
104
105 "no" {yylval.line = yylineno;return NO;}
106
107 "+" { yylval.line = yylineno; sprintf(yylval.sval,"%s",yytext); return MAS; }
108
109 "-" { yylval.line = yylineno; sprintf(yylval.sval,"%s",yytext); return MENOS; }
110
111 "*" { yylval.line = yylineno; sprintf(yylval.sval,"%s",yytext); return PROD; }
112
113 "/" { yylval.line = yylineno; sprintf(yylval.sval,"%s",yytext); return DIV; }
114
115 "%" { yylval.line = yylineno; sprintf(yylval.sval,"%s",yytext); return MOD; }
116
117 "(" {yylval.line = yylineno;return LPAR; }
118
119 ")" {yylval.line = yylineno;return RPAR;}
120
121 ":@" {yylval.line = yylineno;return ASIG;}
122
123 "," {yylval.line = yylineno;return COMA;}
124
125 "[" {yylval.line = yylineno;return LCOR;}
126
127 "]" {yylval.line = yylineno;return RCOR;}
128
129 ";" {yylval.line = yylineno;return PYC;}
130
131 "." {yylval.line = yylineno;return PT;}
132
133 "<" { yylval.line = yylineno; sprintf(yylval.sval,"%s",yytext); return SMT; }
134
135 ">" { yylval.line = yylineno; sprintf(yylval.sval,"%s",yytext); return GRT; }
136
137 ">=" { yylval.line = yylineno; sprintf(yylval.sval,"%s",yytext); return GREQ; }
138
139 "<=" { yylval.line = yylineno; sprintf(yylval.sval,"%s",yytext); return SMEQ; }
140
141 "<>" { yylval.line = yylineno; sprintf(yylval.sval,"%s",yytext); return DIF; }
142
143 "==" { yylval.line = yylineno; sprintf(yylval.sval,"%s",yytext); return EQEQ; }
144
145 {id} { yylval.line = yylineno; sprintf(yylval.sval,"%s",yytext);return ID;}
146
147 [ \n\t\r]+ {}
148
149
150
151 "/*" { BEGIN(com1); }
152
153 <com1>\n { yylval.line = yylineno; }
154
155 <com1>[^*\n]* {}
156
157 <com1>"*"+[^\n]* {}
158
159 <com1>"*"+"/" { BEGIN(INITIAL); }
160
161 "//" { BEGIN(com2); }
162
163 <com2>[^*\n]* {}
164
165 <com2>\n { BEGIN(INITIAL); }
166
167 . { printf("\n****Error lexico en la linea: %d\n", yylineno);}
168
169 %%

```

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.



```
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

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) $L_V \rightarrow L_V1, \text{id}$	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) $L \rightarrow \text{id}$	Si !TS.existe(id) Entonces STS.getCima().append(id, dir, Tipo, 'var', nulo, -1)

	dir \leftarrow 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) $L_ARG \rightarrow T \text{ id } ARG$	L_ARG.lista = nuevaLista() L_ARG.lista.append(T.tipo) L_ARG.num = L_ARG1.num +1
11) $ARG \rightarrow T_ARG \text{ id}$	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) $T_ARG \rightarrow B P_A$	B = B.tipo T_ARG.tipo = P_A.tipo
13) $P_A \rightarrow [] P_A1$	P_A.tipo = STT.append("array",- ,P_A1.tipo, null)
13) $P_A \rightarrow \epsilon$	P_A.tipo = B
14) $S \rightarrow S1S2$	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) $E_B \rightarrow E_B1 \ \&\& \ E_B2$	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) $E_B \rightarrow ! \ E_B1$	B.truelist = B1.falselist B.falselist = B1.truelist B.codigo = B1.codigo
18) $E_B \rightarrow E1 \ R \ E2$	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) $E_B \rightarrow \text{verdadero}$	t0 = nuevoIndice() E_B.truelist = nuevaLista(t0) E_B.codigo = gen('goto' t0)
18) $C \rightarrow \text{falso}$	t0 = nuevoIndice() E_B.falselist = crearLista(t0) E_B.codigo = gen('goto' t0)
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) $R \rightarrow R1 > R2$	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) $V \rightarrow id \ V_C$	
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) $D_S_T \rightarrow D_S_T.id$	
23) $D_S_T \rightarrow \epsilon$	
24) $A \rightarrow id \ [E]$	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

De las anteriores reglas las que utilizaremos en el programa son:

1	$P \rightarrow DF$
2	$D \rightarrow T \ L_V \mid \epsilon$
3	$B \rightarrow ent \mid real \mid dreal \mid car \mid sin \mid struct \{D\}$
4	$L_V \rightarrow L_V1, id \mid id$
5	$T_A \rightarrow [num] \ T_A1 \mid \epsilon$
6	$F \rightarrow def \ T \ id \ (ARG) \ \{DS\} \ F \mid \epsilon$
7	$ARG \rightarrow L_ARG \mid \epsilon$
8	$L_ARG \rightarrow L_ARG1, T \ id_ARG \mid T_ARG \ id$
9	$P_A \rightarrow [] \ P_A1 \mid \epsilon$
10	$S \rightarrow S1S2 \mid si \ (E_B) \ entonces \ S1 \ fin \mid si \ (E_B) \ entonces \ S1 \ sino \ S2 \ fin \mid$ mientras (E_B) hacer $S1 \ fin \mid V := E; \mid devolver \ E; \mid devolver; \mid \{S\} \mid segun \ (V)$ hacer $CP \ fin \mid terminar; \mid escribir \ E;$
11	$CASS \rightarrow caso \ num: \ S \ CASS1$
12	$PRED \rightarrow PRED: \ S \mid \epsilon$
13	$A \rightarrow id \ [E] \mid A1 \ [E]$
14	$E \rightarrow E1 + E2 \mid E1 - E2 \mid E1 * E2 \mid E1 / E2 \mid E1 \% E2 \mid V \mid cadena \mid num \mid car$

15	$P \rightarrow \epsilon \mid L_P$
16	$L_P \rightarrow L_P1, E$
17	$E_B \rightarrow E_B1 \mid E_B2 \mid E_B1 \&\& E_B2 \mid ! E_B1 \mid E1 R E2 \mid \text{verdadero} \mid \text{falso}$
18	$R \rightarrow R1 < R2 \mid R1 > R2 \mid R1 >= R2 \mid R1 <= R2 \mid R1 != R2 \mid R1 = R2$

Los siguientes árboles fueron generados para llevar a cabo el análisis sintáctico

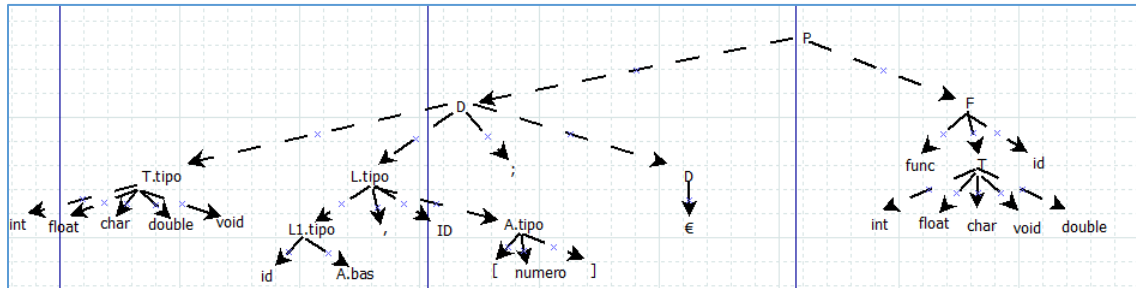


Ilustración 1: Árbol 1

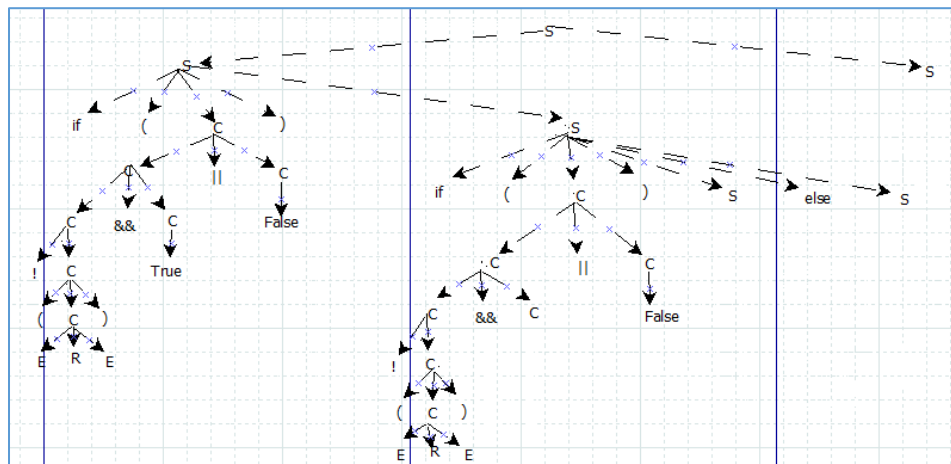


Ilustración 2: Árbol

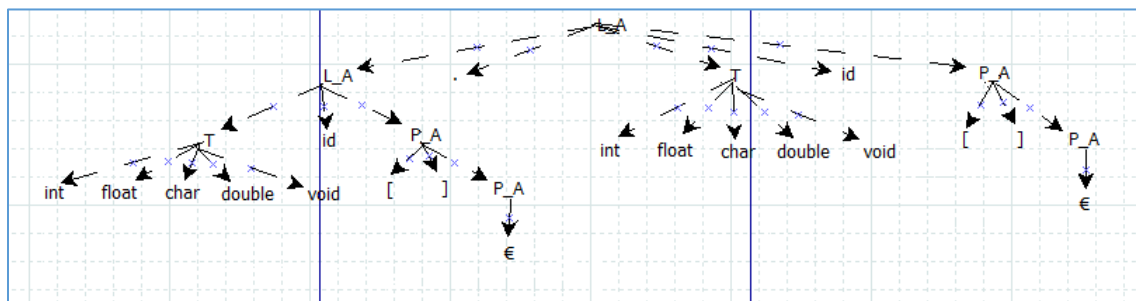
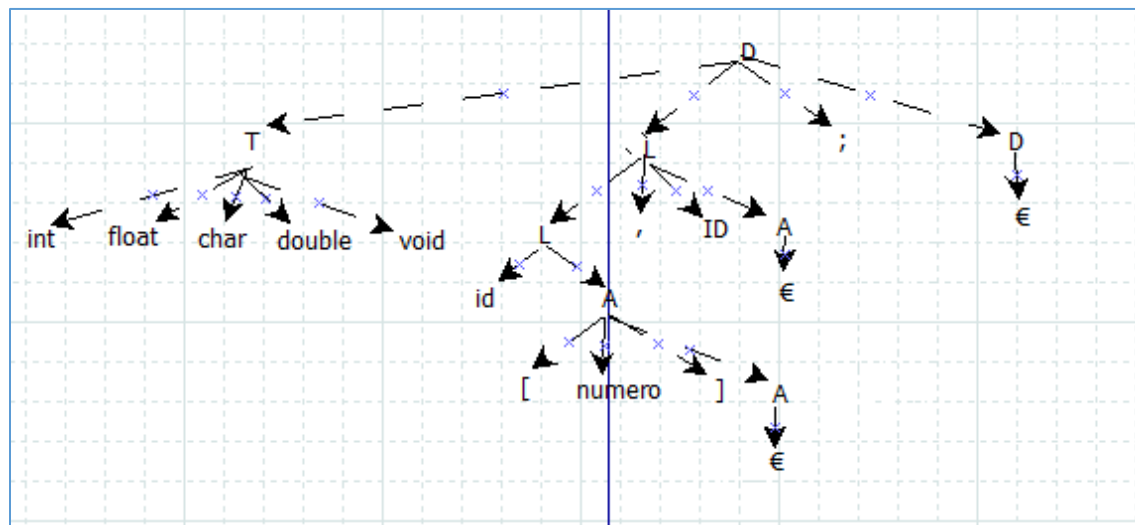


Ilustración 3: Árbol 3



Implementación:

```

1  %{
2  #include <stdio.h>
3  #include <string.h>
4  #include "intermediate_code.h"
5  #include "tablaSimbol.h"
6  #include "tablaTipo.h"
7  #include "pilaTablaSimbol.h"
8  #include "pilaTablaTipo.h"
9
10 int max(int a, int b);
11 int min(int a, int b);
12 void ampliar(char* res, char* dir, int a, int b);
13 void reducir(char* res, char* dir, int a, int b);
14 void yyerror(char *s);
15 extern int yylex();
16 void nuevaEtiqueta(char *dire);
17 int temp, temp2;
18 char temp_E[32];
19 char tipos[32];
20 pilaTipos *ptt;
21 pilaSimbolos *pts;
22 int tipoGlobal, baseGlobal, dirGlobal;
23
24 %}
25
26 %union
27 {
28     char dir[64];
29     int base;
30     int line;
31     struct
32     {
33         int tipo;
34         int ival;
35         float fval;
36         float dval;
37     } num;
38
39     struct
40     {
41         char sval[10];
42         int tipo;
43     } car;
44
45     struct
46     {
47         char dir[24];
48         int tipo;
49     } variaexp;
50 }
51
52 %token PYC
53 %token REGISTRO
54 %token INICIO
55 %token FIN
56 %token ESTRUCTURA
57 %token DEF
58 %token SI
59 %token ENTONCES
60 %token MIENTRAS
61 %token HACER
62 %token SEGUN
63 %token ESCRIBIR
64 %token DEVOLVER
65 %token LEER
66 %token TERMINAR
67 %token CASO
68 %token PRED
69 %token DOSP
70 %token FALSO
71 %token VERDADERO
72
73 %token<base> ENT
74 %token<base> CAR
75 %token<base> DREAL
76 %token<base> REAL
77 %token<base> SIN
78 %token<dir> ID
79 %token<car> CHARACTER
80 %token<dir> CADENA
81 %token<num> NUM
82
83 %token COMA
84 %right ASIG
85 %left O
86 %left Y
87 %left<car> SMT
88 %left<car> GRT
89 %left<car> GREQ
90 %left<car> SMEQ
91 %left<car> DIF
92 %left<car> EQEQ
93 %left<dir> SUM
94 %left<dir> RES
95 %left<dir> MUL
96 %left<dir> DIV

```

```

96 %left<dir> DIV
97 %left<dir> MOD
98 %right NO
99 %nonassoc PT
100 %nonassoc LCOR
101 %nonassoc RCOR
102 %nonassoc LPAR RPAR
103 %nonassoc SINO
104 %type<dir> programa declaraciones tipo_registro lista_var arreglo funciones argumentos lista_arg arg tipo_arg param_arr sentencias sentencia casos predeterminado e_bool relacional
105 dato_est_sin_parametros lista_param_variable_comp A/*No terminales*/
106 %type<base> base tipo tipo_arreglo
107 %type<variaexp> expresion variable
108 %start programa /*Inicio*/
109
110
111 programa : {
112     ptt = crearPilaTipos();
113     pts = crearPilaSimbolos();
114     dirGlobal=0;
115     insertarTablaTipo(ptt,crearTablaTipo());
116     insertarTipo(getTipoCima(ptt),crearTipo("car", 1, -1, 1,false,NULL));
117     insertarTipo(getTipoCima(ptt),crearTipo("ent", 4, -1, 1,false,NULL));
118     insertarTipo(getTipoCima(ptt),crearTipo("real", 4, -1, 1,false,NULL));
119     insertarTipo(getTipoCima(ptt),crearTipo("dreal", 8, -1, 1,false,NULL));
120     insertarTablasimbolo(pts, crearTablasimbolo());
121
122 } declaraciones funciones { printf("No hay errores, la gramática acepto el programa\n");
123 printTablaTipo(getTipoCima(ptt));
124 printTablasimbolos(getSimboloCima(pts));};
125
126 declaraciones : tipo {tipoGlobal = $1;} lista_var PYC declaraciones {}
127 | tipo_registro lista_var PYC declaraciones {}
128 | {};
129
130 tipo_registro : ESTRUCTURA INICIO declaraciones FIN {};
131
132 tipo : base {baseGlobal = $1;} tipo_arreglo {$$ = $3;};
133
134 base : ENT {$$=1;}
135 | REAL {$$=2;}
136 | DREAL {$$=3;}
137 | SIN {$$=-1;}
138 | CAR {$$=0;};
139
140 tipo_arreglo : LCOR NUM RCOR tipo_arreglo {
141     if($2.tipo == 1)
142     {
143         if($2.ival > 0)
144         {
145             $$ = insertarTipo(getTipoCima(ptt),crearTipo("array", getTamanio(getTipoCima(ptt),$4),$4,$2.ival,false,NULL));
146         }
147         else
148         {
149             yyerror("El tamaño del arreglo no es válido\n");
150         }
151     }
152     else
153     {
154         yyerror("El tamaño del arreglo no es un entero\n");
155     }
156 }
157 | {$$ = baseGlobal;};
158
159 lista_var : ID A {
160     if(buscar(getSimboloCima(pts), $1) == -1)
161     {
162         insertar(getSimboloCima(pts),crearSimbolo($1, tipoGlobal, dirGlobal, "var"));
163         dirGlobal = dirGlobal + getTamanio(getTipoCima(ptt),tipoGlobal);
164     }
165     else
166     {
167         yyerror("El identificador ya fue declarado\n");
168     }
169 };
170
171 A : COMA ID A {
172     if(buscar(getSimboloCima(pts), $2) == -1)
173     {
174         insertar(getSimboloCima(pts),crearSimbolo($2, tipoGlobal, dirGlobal, "var"));
175         dirGlobal = dirGlobal + getTamanio(getTipoCima(ptt),tipoGlobal);
176     }
177     else
178     {
179         yyerror("El identificador ya fue declarado\n");
180     }
181 }
182 | {};
183
184 funciones : DEF tipo ID LPAR argumentos RPAR INICIO declaraciones sentencias FIN funciones {}
185 | {};
186
187 argumentos : lista_arg {}
188 | SIN {};
189
190 lista_arg : lista_arg COMA arg {}

```



```

190 lista_arg : lista_arg COMA arg {}
191 | arg {};
192
193 arg : tipo_arg ID {};
194
195 tipo_arg : base param_arr {};
196
197 param_arr : LCOR RCOR param_arr {}
198 | {};
199
200 sentencias : sentencias sentencia {}
201 | {};
202
203 sentencia : SI e_bool ENTONCES sentencias FIN {}
204 | SI e_bool ENTONCES sentencias SINO sentencias FIN {}
205 | MIENTRAS e_bool HACER sentencias FIN {}
206 | HACER sentencias MIENTRAS e_bool PYC {}
207 | SEGUN LPAR variable RPAR HACER casos predeterminado FIN {}
208 | variable ASIG expresion PYC {
209     char* resultado = (char*)malloc(sizeof(char)*10);
210     reducir(resultado,$3.dir,$3.tipo,$1.tipo);
211     printf("%s = %s\n",$1.dir,resultado);
212 }
213 | ESCRIBIR expresion PYC {}
214 | LEER variable PYC {}
215 | DEVOLVER PYC {}
216 | DEVOLVER expresion PYC {}
217 | TERMINAR PYC {}
218 | INICIO sentencias FIN {};
219
220 casos : CASO NUM DOSP sentencias casos {}
221 | CASO NUM DOSP sentencias {};
222
223 predeterminado : PRED DOSP sentencias {}
224 | {};
225
226 e_bool : e_bool 0 e_bool {}
227 | e_bool Y e_bool {}
228 | NO e_bool {}
229 | relacional {}
230 | VERDADERO {}
231 | FALSO {}
232
233 relacional : relacional SMT relacional {}
234 |relacional GRT relacional {}
235 |relacional GREQ relacional {}
236 |relacional SMEQ relacional {}
237 |relacional DIF relacional {}
238 |relacional DIF relacional {}
239 |relacional EQEQ relacional {}
240 | expresion {};
241
242 ▼ expresion : expresion SUM expresion {
243     $$.tipo = max($1.tipo,$3.tipo);
244     char* temporal = (char*)malloc(sizeof(char)*10);
245     newTemp(temporal);
246     char* temporal1 = (char*)malloc(sizeof(char)*10);
247     char* temporal2 = (char*)malloc(sizeof(char)*10);
248     ampliar(temporal1,$1.dir,$1.tipo,$3.tipo);
249     ampliar(temporal2,$3.dir,$3.tipo,$1.tipo);
250     printf("%s = %s + %s\n",temporal,temporal1,temporal2);
251     strcpy($$.dir,temporal);
252 }
253 ▼ expresion RES expresion{
254     $$.tipo = max($1.tipo,$3.tipo);
255     char* temporal = (char*)malloc(sizeof(char)*10);
256     newTemp(temporal);
257     char* temporal1 = (char*)malloc(sizeof(char)*10);
258     char* temporal2 = (char*)malloc(sizeof(char)*10);
259     ampliar(temporal1,$1.dir,$1.tipo,$3.tipo);
260     ampliar(temporal2,$3.dir,$3.tipo,$1.tipo);
261     printf("%s = %s - %s\n",temporal,temporal1,temporal2);
262     strcpy($$.dir,temporal);
263 }
264 ▼ expresion MUL expresion {
265     $$.tipo = max($1.tipo,$3.tipo);
266     char* temporal = (char*)malloc(sizeof(char)*10);
267     newTemp(temporal);
268     char* temporal1 = (char*)malloc(sizeof(char)*10);
269     char* temporal2 = (char*)malloc(sizeof(char)*10);
270     ampliar(temporal1,$1.dir,$1.tipo,$3.tipo);
271     ampliar(temporal2,$3.dir,$3.tipo,$1.tipo);
272     printf("%s = %s * %s\n",temporal,temporal1,temporal2);
273     strcpy($$.dir,temporal);
274 }
275 ▼ expresion DIV expresion{
276     $$.tipo = max($1.tipo,$3.tipo);
277     char* temporal = (char*)malloc(sizeof(char)*10);
278     newTemp(temporal);
279     char* temporal1 = (char*)malloc(sizeof(char)*10);
280     char* temporal2 = (char*)malloc(sizeof(char)*10);
281     ampliar(temporal1,$1.dir,$1.tipo,$3.tipo);
282     ampliar(temporal2,$3.dir,$3.tipo,$1.tipo);
283     printf("%s = %s / %s\n",temporal,temporal1,temporal2);
284     strcpy($$.dir,temporal);

```

```

285 | expression MOD expression{
286     $$tipo = max($1.tipo,$3.tipo);
287     char* temporal = (char*)malloc(sizeof(char)*10);
288     newTemp(temporal);
289     char* temporal1 = (char*)malloc(sizeof(char)*10);
290     char* temporal2 = (char*)malloc(sizeof(char)*10);
291     ampliar(temporal1,$1.dir,$1.tipo,$3.tipo);
292     ampliar(temporal2,$3.dir,$3.tipo,$1.tipo);
293     printf("%s = %s o/o %s\n",temporal,temporal1,temporal2);
294     strcpy($$.dir,temporal);
295 }
296 | LPAR expression RPAR {strcpy($$.dir,$2.dir);
297     $$tipo = $2.tipo;
298 }
299 | variable {strcpy($$.dir,$1.dir);
300     $$tipo = $1.tipo;
301 }
302 | NUM {sprintf($$.dir,"%d",$1.ival);
303     $$tipo = $1.tipo;
304 }
305 }
306 | CADENA {}
307 | CARACTER {};
308
309 variable : ID variable_comp {strcpy($$.dir,$1); $$tipo = 1;};
310
311 variable_comp : dato_est_sim {}
312 | arreglo {}
313 | LPAR parametros RPAR {};
314
315 dato_est_sim : dato_est_sim PT ID {}
316 | {};
317
318 arreglo : LCOR expression RCOR {}
319 | LCOR expression RCOR arreglo {};
320
321 parametros : lista_param {}
322 | {};
323
324 lista_param : lista_param COMA expression {}
325 | expression {};
326 %%
327
328 void yyerror(char *s){
329     printf("Error sintactico. %s\n",s);
330 }
331
332

```

```

333 void nuevaEtiqueta(char *dire){
334     char L[32];
335     sprintf(L,"L%d", temp2++);
336     strcpy (dire,L);
337 }
338
339 int max(int a, int b)
340 {
341     if(a < 3 && b < 3 || a==b)
342     {
343         if(a > b)
344         {
345             return a;
346         }
347         else
348         {
349             return b;
350         }
351     }
352     else
353     {
354         yyerror("Tipos incompatibles\n");
355     }
356 }
357
358 int min(int a, int b)
359 {
360     if(a < 3 && b < 3 || a==b)
361     {
362         if(a < b)
363         {
364             return a;
365         }
366         else
367         {
368             return b;
369         }
370     }
371     else
372     {
373         yyerror("Tipos incompatibles\n");
374     }
375 }
376
377 void ampliar(char* res,char* dir,int a, int b)
378 {
379     if(a != b)
380     {

```

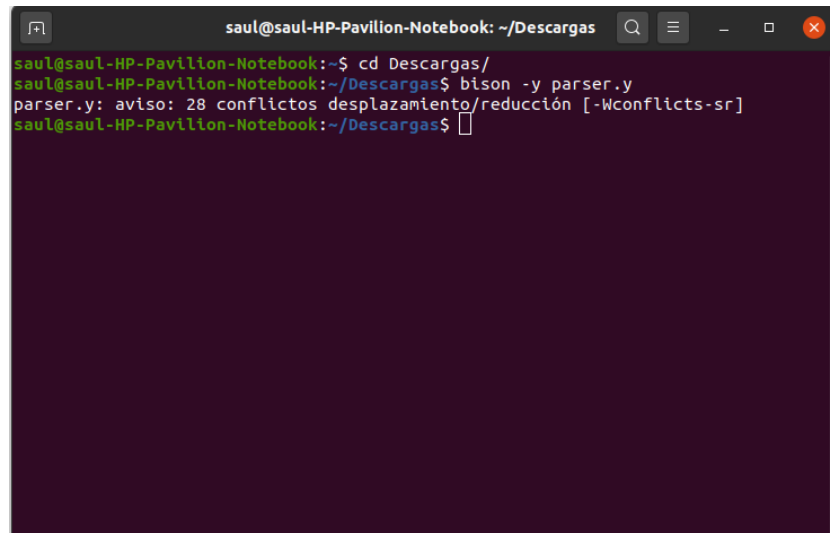
```

381         int m = max(a,b);
382         if(m == a)
383         {
384             strcpy(res,dir);
385         }
386         else
387         {
388             newTemp(res);
389             switch(m)
390             {
391                 case 1:
392                     printf("%s = (ent)%s\n", res,dir);
393                     break;
394                 case 2:
395                     printf("%s = (real)%s\n", res,dir);
396                     break;
397                 case 3:
398                     printf("%s = (dreal)%s\n", res,dir);
399                     break;
400                 default:
401                     res = NULL;
402             }
403         }
404     }
405     else
406     {
407         strcpy(res,dir);
408     }
409 }
410
411 void reducir(char* res,char* dir,int a, int b)
412 {
413     if(a != b)
414     {
415         int m = min(a,b);
416         if(m == a)
417         {
418             strcpy(res,dir);
419         }
420         else
421         {
422             newTemp(res);
423             switch(m)
424             {
425                 case 1:
426                     printf("%s = (ent)%s\n", res,dir);
427                     break;
428                 case 2:
429                     printf("%s = (real)%s\n", res,dir);
430                     break;
431                 case 0:
432                     printf("%s = (car)%s\n", res,dir);
433                     break;
434                 default:
435                     res = NULL;
436             }
437         }
438     }
439     else
440     {
441         strcpy(res,dir);
442     }
443 }

```

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 -d -v "nombre.y"*.

A terminal window titled 'saul@saul-HP-Pavilion-Notebook: ~/Descargas' with search, menu, and window control icons. The terminal shows the following commands and output:

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

Compilación de todo el programa:

1. Para compilar el programa completo debemos de ir al fichero donde tengamos a todos nuestros archivos.
2. Después de eso ejecutamos los dos pasos anteriores.
3. Ejecutamos el siguiente código: `gcc -o prueba.o *.h *.c`
4. Después de eso ejecutamos: `./Prueba.txt`

Y saldrá lo siguiente:

Error sintactico. El identificador ya fue declarado

Error sintactico. El identificador ya fue declarado

Error sintactico. El identificador ya fue declarado

```
x = u
t0 = 1 + 2
f = t0
t1 = 8 * 6
t2 = 9 - t1
t3 = 10 + t2
t4 = 100 * 20
t5 = t3 / t4
h = t5
t6 = h * 10
t7 = f / t6
t8 = x + t7
z = t8
t9 = c * 12
t10 = b * t9
a = t10
t11 = a + e
t12 = t11 + f
t13 = t12 + g
t14 = t13 + h
d = t14
```

No hay errores, la gramática acepto el programa

#####TABLA DE TIPOS#####

ID	NOMBRE	TAM	TIPO	BASE
0	car	1	-1	
1	ent	4	-1	
2	real	4	-1	
3	dreal	8	-1	
4	array	20	1	

*****TABLA DE SIMBOLOS*****

NUM	ID	TIPO	DIR	TipoVar	Params
1	z	1	0	var	
2	h	1	4	var	
3	f	1	8	var	
4	x	1	12	var	
5	g	3	16	var	
6	e	3	24	var	
7	d	3	32	var	
8	c	3	40	var	
9	b	3	48	var	
10	a	3	56	var	
11	il	0	64	var	
12	ch	0	65	var	
13	pp	2	66	var	
14	hola	3	70	var	

ulysses@Odisseo:~/Descargas/Compiladores\$

El código de prueba es el siguiente:

```
1
2  ent x,f,h,z;
3  dreal a,b,c,d,e,f,g,h;
4  car ch,il;
5  real pp;
6  dreal hola;
7  ent [5] a;
8  def ent main(ent UNO)
9  ▼ inicio
10     x := u;
11     f := 1 + 2;
12     h := (10 + 9 - 8 * 6) / (100 * 20);
13     z := x + f / (h * 10);
14     a := b * (c * 12);
15     d := a + e + f + g + h;
16  fin
```

Main.c

```
1  /*
2     Equipo de trabajo:
3     -Álvarez González Ian
4     -García Oviedo Jaasiel Osmar
5     -López López Ulysses
6     -Domínguez Cisneros Alexis Saúl
7     Materia: Compiladores
8     Grupo: 01
9     Semestre: 2020-1
10    Fecha última versión: 13 de junio de 2020
11  */
12  #include<stdio.h>
13  extern int yyparse();
14  extern FILE *yyin;
15
16  int main(int argc, char** argv)
17  {
18      FILE *f;
19      if(argc<2)
20      {
21          printf("Faltan argumentos\n");
22          return -1;
23      }
24      f= fopen(argv[1],"r");
25      if(!f)
26      {
27          printf("El archivo %s no se puede abrir\n", argv[1]);
28          return -1;
29      }
30      yyin= f;
31      yyparse();
32      fclose(yyin);
33      return 0;
34  }
```