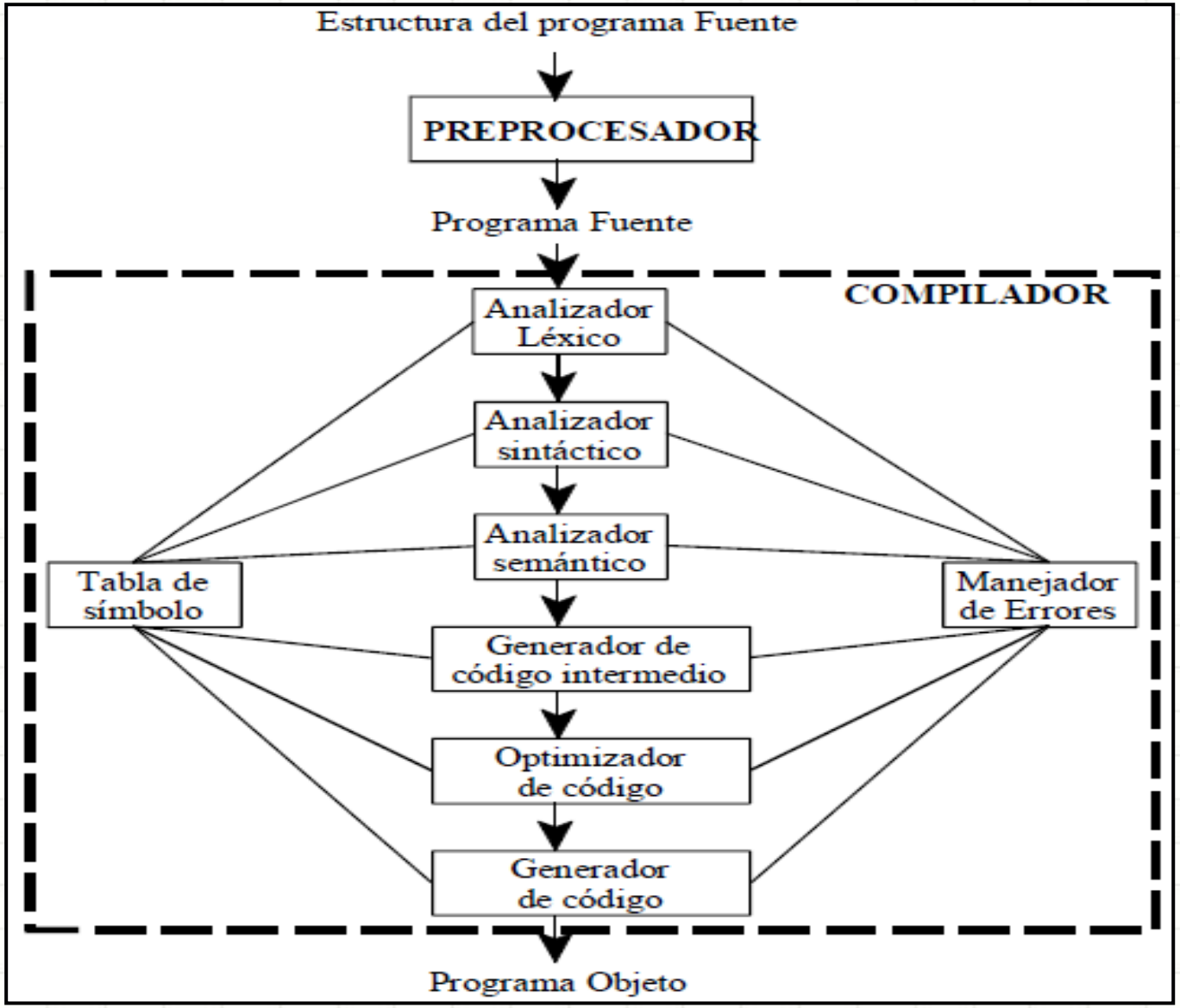
COMPILADORES
TEMA 4. ANÁLISIS SEMÁNTICO

Presenta: Mtro. David Martínez Torres
Universidad Tecnológica de la Mixteca
Instituto de Computación
Oficina No. 37
dtorres@gs.utm.mx

Contenido

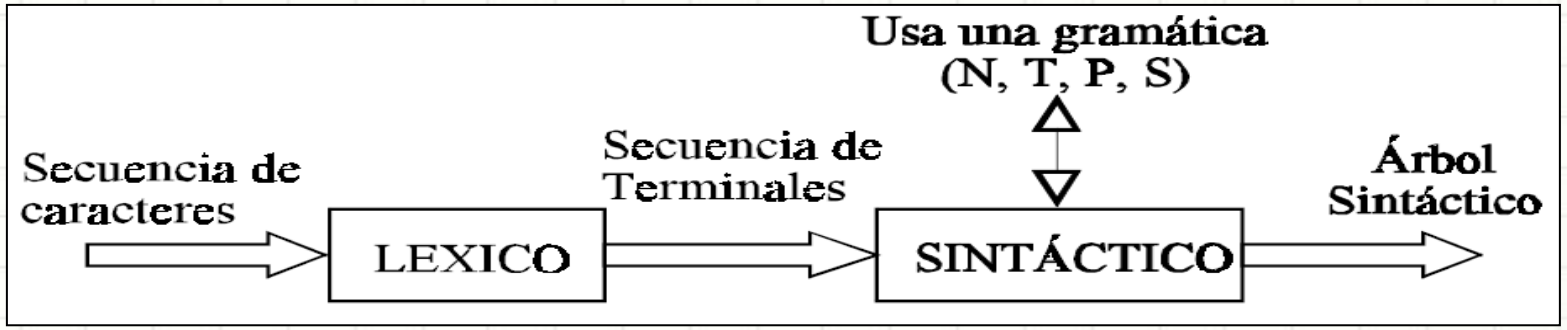
1. Funcionalidad del analizador semántico
2. Traducción dirigida por la sintaxis
3. Gramática de atributos
4. Comprobaciones semánticas
5. Comprobación de tipos
6. Generadores de analizadores léxicos, sintácticos y semánticos.

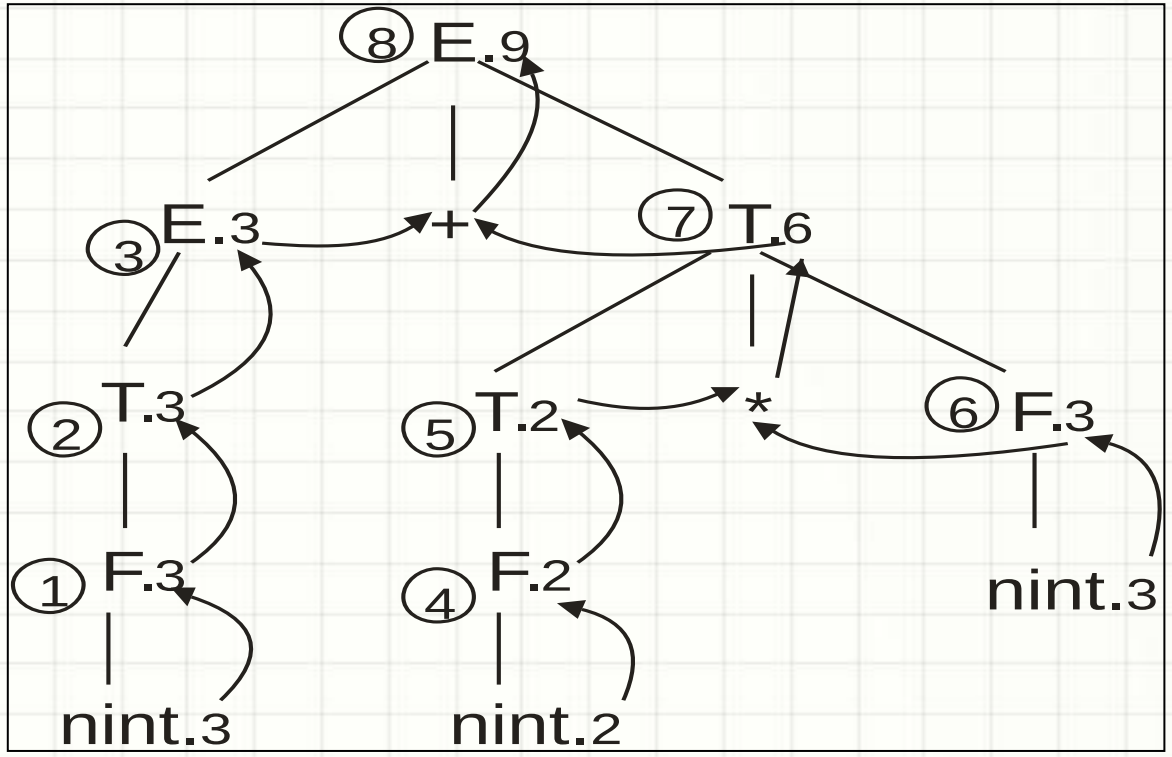
1. Funcionalidad del analizador semántico.



1. Funcionalidad del analizador semántico.

Recordando la interacción entre las fases del compilador.





1. Funcionalidad del analizador semántico

- Además de comprobar que un programa esté sintácticamente bien escrito, se tiene que comprobar que lo que se quiere hacer tiene sentido.
- El análisis semántico dota de un significado coherente a lo que se ha hecho en el análisis sintáctico.
- El chequeo semántico se encarga de que los tipos estén correctos. Por ejemplo, no podemos multiplicar una cadena de caracteres por un entero.

1. Funcionalidad del analizador semántico

Análisis semántico: Comprueba todo lo relacionado con el significado: chequeo de tipos, rangos de valores, existencia de variables, etc.

Llamadas a función con diferentes número y tipos en los parámetros, diferentes tipos de devolución, etc.

Ejemplos:

int num, num, suma=0 ;
float numreal ;

suma= numreal ;

Fijarse en la redeclaración de variables y en la asignación de diferentes tipos, son errores semánticos.

Ejemplo 1. Análisis semántico.

Recordando el ejemplo del análisis léxico y sintáctico de la gramática de expresiones aritméticas, ahora en análisis sintáctico-semántico LR, se diseñará una calculadora como un ejemplo general.

Gramática

1. E→E+T
2. E→T
3. T→T*F
4. T→F
5. F→(E)
6. F→nint

Ejemplo 1. Análisis semántico.

1. Considere como ejemplo el programa fuente.

   3 + 2 * 3

2. Tira de tokens obtenida con el análisis léxico:

   nint + nint * nint

3. Resultado esperado de la evaluación de la expresión con el análisis semántico:

   3+2*3 = **9**

Gramática

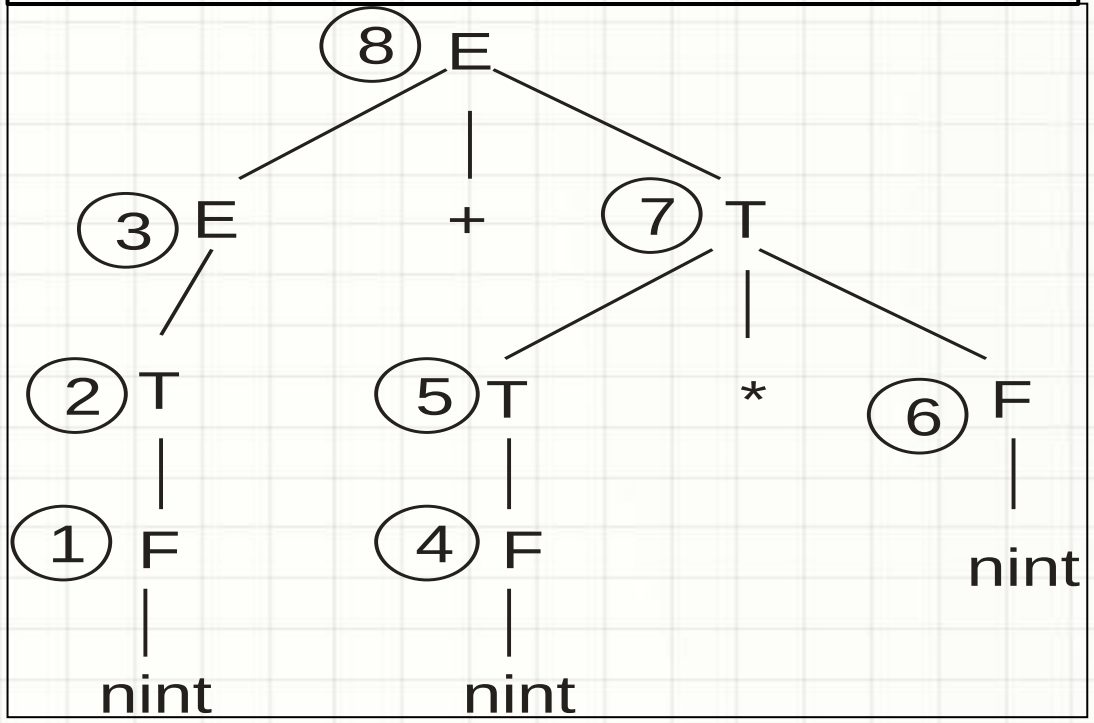
1. E→E+T
2. E→T
3. T→T*F
4. T→F
5. F→(E)
6. F→nint

Ejemplo 1. Análisis semántico.

Resultado del análisis sintáctico ascendente LR

PILA	ENTRADA	ACCION
0	nint + nint * nint $	d5
0 nint 5	+ nint * nint $	r6 F→nint
0 F 3	+ nint * nint $	r4 T→F
0 T 2	+ nint * nint $	r2 E→T
0 E 1	+ nint * nint $	d6
0 E 1 + 6	nint * nint $	d5
0 E 1 + 6 nint 5	* nint $	r6 F→nint
0 E 1 + 6 F 3	* nint $	r4 T→F
0 E 1 + 6 T 9	* nint $	d7
0 E 1 + 6 T 9 * 7	nint $	d5
0 E 1 + 6 T 9 * 7 nint 5	$	r6 F→nint
0 E 1 + 6 T 9 * 7 F 10	$	r3 T→T*F
0 E 1 + 6 T 9	$	r1 E→E+T
0 E 1	$	Aceptar

Árbol sintáctico obtenido de la última derivación a la primera.



Ejemplo 1. Análisis semántico.

Para que el **análisis semántico** realice la **traducción**, requerirá que el **analizador léxico**, envíe los valores(datos) al analizador sintáctico.

- Para el programa de ejemplo: 3 + 2 * 3
- Debe enviar la tira de tokens asociada con su valor mediante un atributo **.valorLex**:

  nint.3 + nint.2 * nint.3

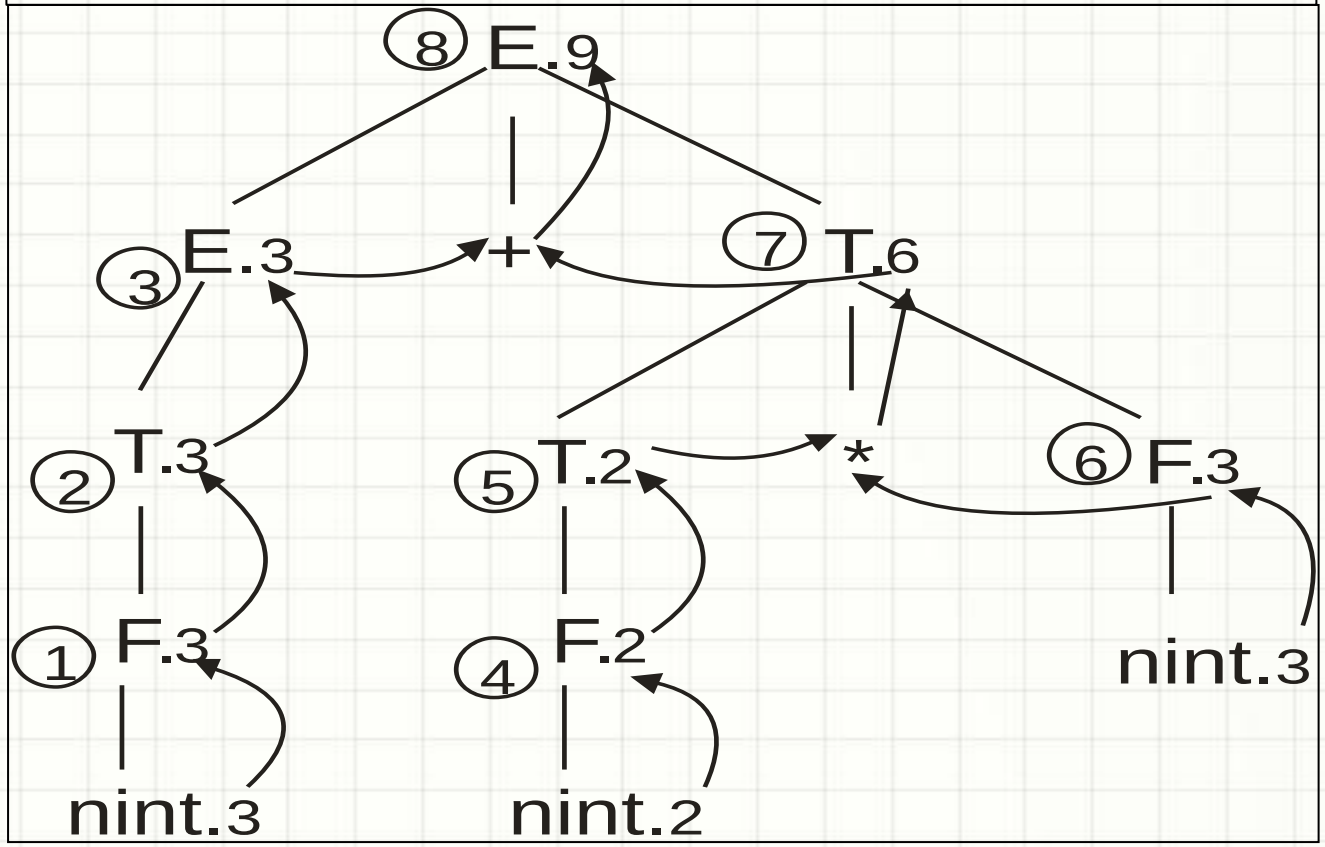
- Parecido a como lo hace la herramienta lex:

  [0-9]+ {yylval = atoi(yytext);
  return nint}

Ejemplo 1. Análisis semántico.

Con esos atributos del léxico, conforme se hace el reconocimiento sintáctico, se puede ir construyendo el resultado a devolver.

Árbol sintáctico adornado que se obtendrá al realizar el análisis sintáctico-semántico



Ejemplo 1. Análisis semántico (DDS)

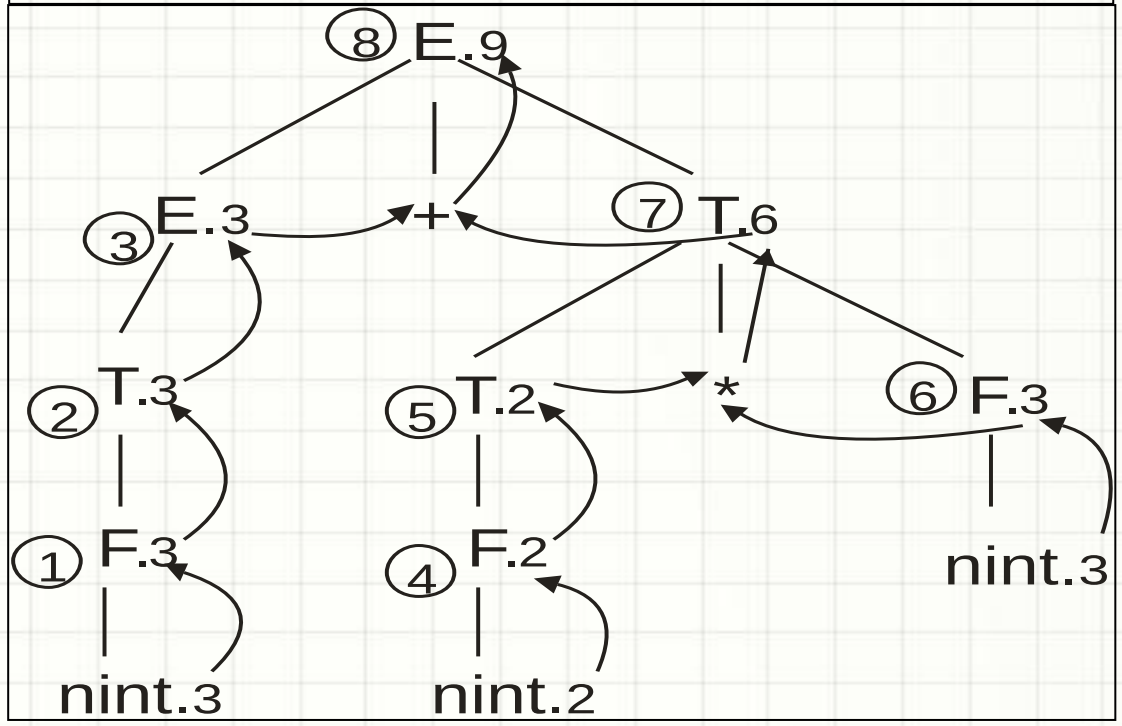
Para esto, se requiere agregar las **acciones semánticas** necesarias a la reglas de producción de la gramática, para generar la traducción definida, a esto se le conoce como DDS(Definición Dirigida por Sintaxis)

Gramática	Reglas semánticas
1. E→E+T	{E_0.v:=E_1.v+T.v; printf("Resultado: %d", E.v)}
2. E→T	{E.v:=T.v}
3. T→T*F	{T_0.v:=T_1.v * F.v}
4. T→F	{T.v:=F.v}
5. F→(E)	{F.v:=E.v}
6. F→nint	{F.v:=nint.valex}

Para el ejemplo de la tira de tokens:

nint.3 + nint.2 * nint.3

Se obtendrá el siguiente árbol de análisis sintáctico-semántico adornado



Ejemplo 1. Análisis semántico (DDS)

Para el análisis semántico, la tabla de análisis sintáctico no cambia.

Edo	Acción						Ir_a		
	id	+	*	(	)	$	E	T	F
0	d5			d4			1	2	3
1		d6				acep			
2		r2	d7		r2	r2			
3		r4	r4		r4	r4			
4	d5			d4			8	2	3
5		r6	r6		r6	r6			
6	d5			d4				9	3
7	d5			d4					10
8		d6			d11				
9		r1	d7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Revise paso a paso como el analizador sintáctico-semántico realiza la traducción.

PILA	ENTRADA	ACCION
0	nint.3 + nint.2 * nint.3 $	d5
0 **nint.3** 5	+ nint.2 * nint.3 $	r6 F→nint {F.v:=nint.valex}
0 **F.3** 3	+ nint.2 * nint.3 $	r4 T→F {T.v:=F.v}
0 **T.3** 2	+ nint.2 * nint.3 $	r2 E→T {E.v:=T.v}
0 **E.3** 1	+ nint.2 * nint.3 $	d6
0 **E.3** 1 + 6	nint.2 * nint.3 $	d5
0 **E.3** 1 + 6 **nint.2** 5	* nint.3 $	r6 F→nint {F.v:=nint.valex}
0 **E.3** 1 + 6 **F.2** 3	* nint.3 $	r4 T→F {T.v:=F.v}
0 **E.3** 1 + 6 **T.2** 9	* nint.3 $	d7
0 **E.3** 1 + 6 **T.2** 9 * 7	nint.3 $	d5
0 **E.3** 1 + 6 **T.2** 9 * 7 **nint.3** 5	$	r6 F→nint {F.v:=nint.valex}
0 **E.3** 1 + 6 **T.2** 9 * 7 **F.3** 10	$	r3 T→T*F {T.v:=T.v*F.v}
0 **E.3** 1 + 6 **T.6** 9	$	r1 E→E+T {E.v:=E.v+T.v}
0 **E.9** 1	$	Aceptar
		Resultado: 9

Ejemplo 2. Análisis semántico (DDS)

A continuación se presenta un nuevo ejemplo de análisis sintáctico-semántico LR para la siguiente gramática que declara lista de variables de C.

Gramática	**Acciones semánticas**
1. D → T id L ;	{D.trad := “var ” ‖ id.valex ‖ L.trad ‖ “:” ‖ T.trad ‖ “;”}
2. L → , id L	{L.trad := “,” ‖ id.valex ‖ L.trad}
3. L → ε	{L.trad := “”}
4. T → float	{T.trad := “real”}
5. T → int	{T.trad := “integer”}

Ejemplo de programa

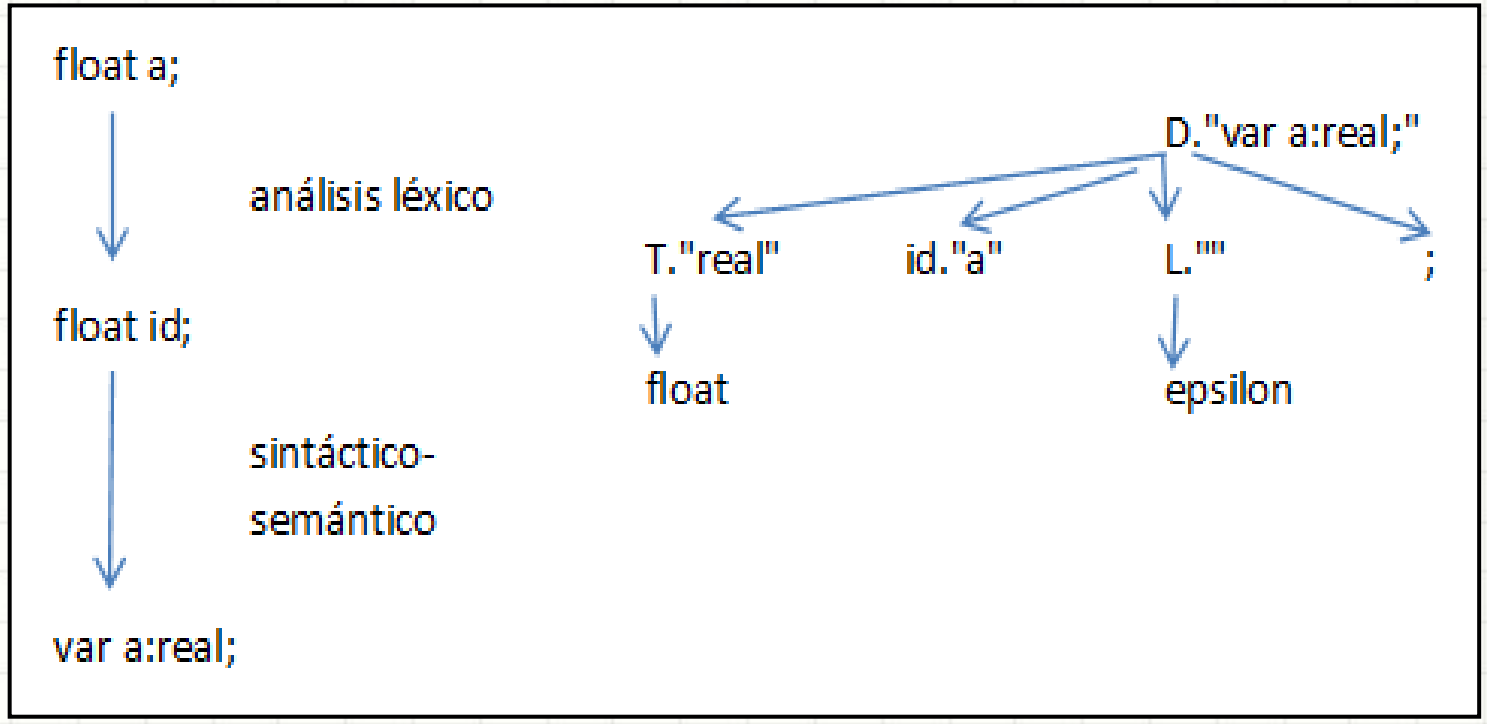
int a, b; **Analizador léxico** int id."a" , id."b" ;

Traducción esperada en lenguaje pascal con el analizador semántico.

var a, b: integer;

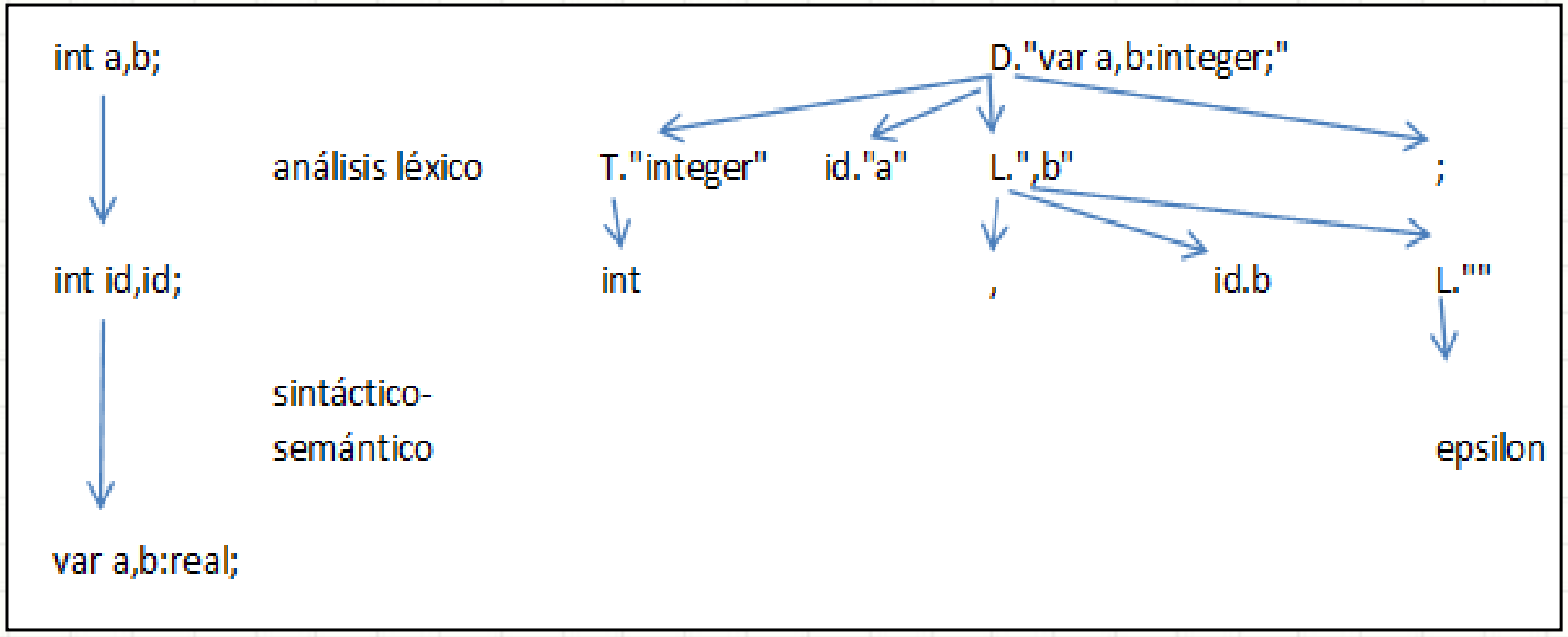
Probando que sí funcionan las acciones semánticas.

Gramática	Acciones semánticas
1. D → T id L ;	{D.trad := “var ” ‖ id.valex ‖ L.trad ‖ “:” ‖ T.trad ‖ “;”}
2. L → , id L	{L.trad := “,” ‖ id.valex ‖ L.trad}
3. L → ε	{L.trad := “”}
4. T → float	{T.trad := “real”}
5. T → int	{T.trad := “integer”}



Otra prueba que sí funcionan las acciones semánticas

Gramática	Acciones semánticas
1. D → T id L ;	{D.trad := “var ” ‖ id.valex ‖ L.trad ‖ “:” ‖ T.trad ‖ “;”}
2. L → , id L	{L.trad := “,” ‖ id.valex ‖ L.trad}
3. L → ε	{L.trad := “”}
4. T → float	{T.trad := “real”}
5. T → int	{T.trad := “integer”}



Ejemplo 2. Análisis semántico (DDS)

- Para el programa de ejemplo:

  int a, b;

- La tira de tokens con sus valores léxicos será la siguiente:

  int id."a" , id."b" ;

Tabla de análisis sintáctico obtenida en el tema 3 es la siguiente:

							Ir_a		
Edo	id	;	,	float	int	$	D	L	T
0				d3	d4		1		2
1						acep			
2	d5								
3	r4								
4	r5								
5		r3	d7					6	
6		d8							
7	d9								
8						r1			
9		r3	d7					10	
10		r2							

Análisis sintáctico-semántico LR

Análisis sintáctico-semántico LR

PILA	ENTRADA	SALIDA
0	int id."a",id."b";$	d4
0int4	id."a",id."b";$	r5 T → int {T.trad := “integer”}
0T."integer"2	id."a",id."b";$	d5
0T."integer"2id."a"5	,id."b";$	d7
0T."integer"2id."a"5,7	id."b";$	d9
0T."integer"2id."a"5,7id."b"9	;$	r3 L → ε {L.trad := “”}
0T."integer"2id."a"5,7id."b"9L.""10	;$	r2 L → , id L {L.trad := “,” || id.valex || L.trad}
0T."integer"2id."a"5L.",b"6	;$	d8
0T."integer"2id."a"5L.",b"6;8	$	r1 D → T id L ; {D.trad := “var ” || id.valex || L.trad || “:” || T.trad || “;”}
0D."var a,b:integer;"1	$	aceptación

Programa fuente en C
int a,b;

Programa destino en pascal
var a,b:integer;

Realice el análisis léxico, sintáctico-semántico para el siguiente ejercicio.

EJEMPLO DE DEFINICION DIRIGIDA POR LA SINTAXIS

int c[10];	var c: array[0..9] of integer;
float d[5][6];	var d:array[0..4,0..5] of real;
char e;	var e:char;
char f[3][5][7];	var f:array[0..2,0..4,0..6] of char;
S→TV;	{S.trad:="var" ‖ V.trad ‖ T.trad ‖ ";"}
V→id V'	{if V'.trad!="" V.temp:="array[" ‖ V'.trad else V.temp:=V'.trad V.trad:=id.valex ‖ ":" ‖ V.temp }
V'→[nint]V'	{if V_1'.trad="" V_0'.trad:="0.." ‖nint.valex-1 ‖ "] of" else V_0'.trad:="0.." ‖nint.valex-1 ‖ "," ‖ V'_1.trad }
V'→ε	{V'.trad:=""}
T→int	{T.trad:="integer"}
T→char	{T.trad:="char"}
T→float	{T.trad:="real"}

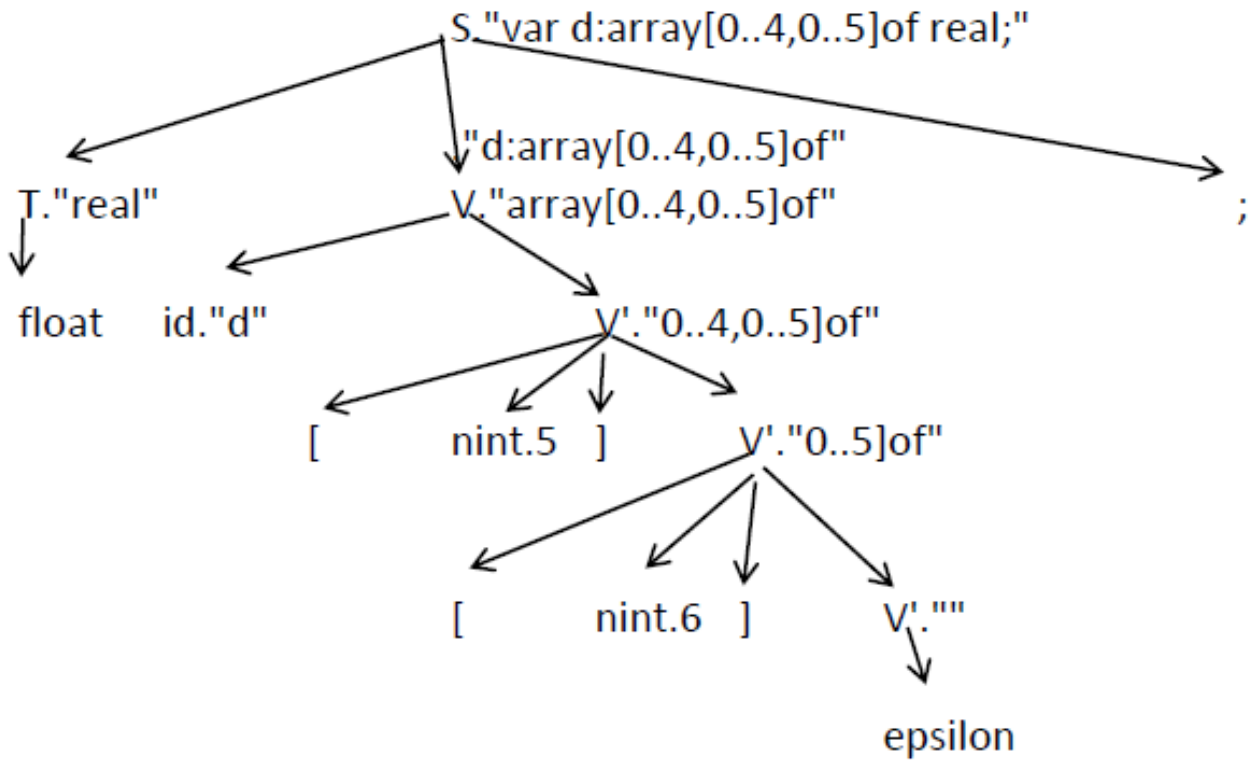
Ejemplo de árbol semántico para gramática que reconoce declaración de arreglos

EJEMPLO DE DEFINICION DIRIGIDA POR LA SINTAXIS

int c[10];	var c: array[0..9] of integer;
float d[5][6];	var d:array[0..4,0..5] of real;
char e;	var e:char;
char f[3][5][7];	var f:array[0..2,0..4,0..6] of char;

S→TV;	{S.trad:=”var” ‖V.trad‖T.trad‖”;”}
V→id V’	{if V’.trad!=”” V.temp:=”array[“‖V’.trad else V.temp:=V’.trad V.trad:=id.valex‖”:”‖V.temp }
V’→[nint]V’	{if V_1’.trad=”” V_0’.trad:=”0..” ‖nint.valex-1‖”] of” else V_0’.trad:=”0..” ‖nint.valex-1‖”,”‖V’$_1$.trad }
V’→ε	{V’.trad:=””}
T→int	{T.trad:=”integer”}
T→char	{T.trad:=”char”}
T→float	{T.trad:=”real”}

float id."d"[nint.5][nint.6];



Versión 2. Acciones semánticas para la misma gramática de declaración de arreglos.

int c[10];	var c:array[0..9] of integer;
float d[5][6];	var d:array[0..4,0..5] of real;
char e;	var e:char;
char f[3][5][7];	var f:array[0..2,0..4,0..6] of char;

1. S → T V ; {S.trad:="var " ‖ V.trad ‖ T.trad ‖ ";"}
2. V → id V′ {if V′.trad!=""
   V.trad:="array[" ‖ V′.trad
   else
   V.trad:=V′.trad
   V.trad:=id.valex ‖ ":" ‖ V.trad
   }
3. V′ → [nint] V′ { V'_0.trad:= "0.." ‖ nint.valex-1
   if V'_1.trad=""
   V'_0.trad:= V'_0.trad ‖ "] of"
   else
   V'_0.trad:= V'_0.trad ‖ "," ‖ V'_1.trad
   }
4. V′ → ε {V′.trad:=""}
5. T → int {T.trad:="integer"}
6. T → char {T.trad:="char"}
7. T → float {T.trad:="real"}

Ejercicio. Análisis semántico (DDS)

Para la siguiente gramática que reconoce una o más listas de declaraciones de variables, diseñe las acciones semánticas para que traduzca al lenguaje pascal. Esta gramática se probará en su analizador semántico.

Gramática

1. D → D D
2. D → T id L ;
3. L → , id L
4. L → ε
5. T → float
6. T → int

Ejemplo 1:

float a;

var a: real;

Ejemplo 2:

int a,b;
float c;

var a,b:integer;
c: real;

Ejemplo 3:

float a;

int b,c;

float d,e;

var a:real;
b,c: integer;
d,e:real;

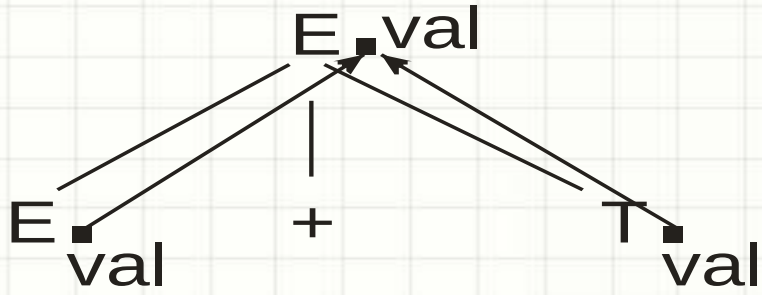
2. Traducción dirigida por la sintaxis (TDS)

- El análisis semántico también es conocido como traducción dirigida por sintaxis (TDS).
- La TDS, requiere de atributos para generar la traducción o evaluación del programa.
- El uso de atributos en una gramática, es conocida como Gramática de atributos.

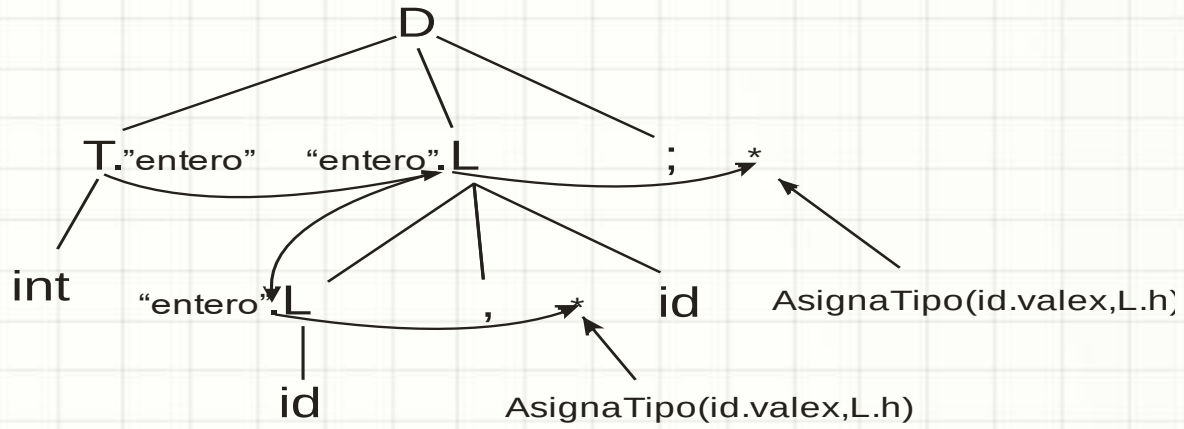
2. Traducción dirigida por la sintaxis (TDS)

Los atributos pueden ser:

- Atributos sintetizados:
  - El valor de un nodo padre se recibe de valores de atributos de símbolos hijos en el árbol de análisis sintáctico.



- Atributos heredados:
  - El valor de un nodo se recibe de valores de atributos de símbolos hermanos y padre de dicho nodo. Ej. para L.



2. Traducción dirigida por la sintaxis (TDS)

Métodos de traducción:

- La **TDS** es una gramática libre de contexto, donde no es necesario que el usuario especifique explícitamente el orden en que tiene lugar la traducción.

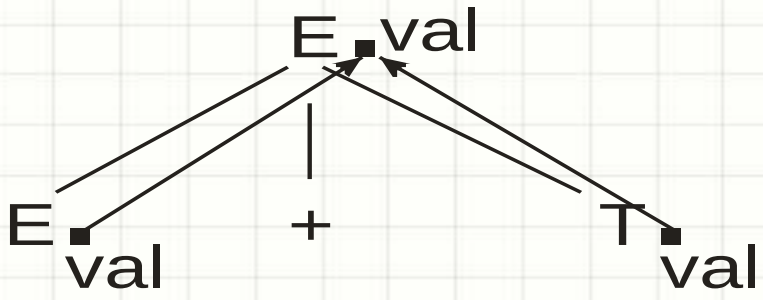
$$A \rightarrow cdB \ \{printf(c + d);\}$$

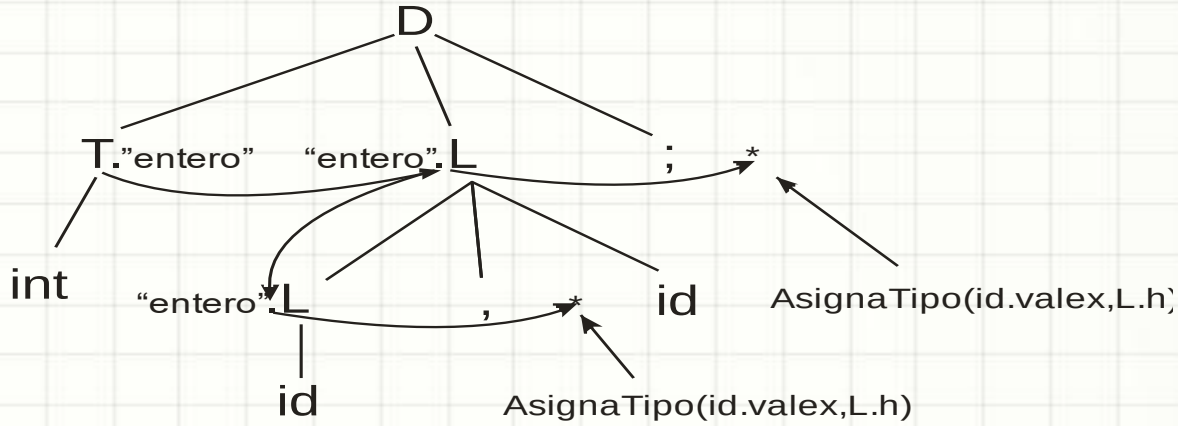
- **Esquema de traducción**: También es una gramática libre de contexto en la que se indica el orden en que se deben evaluar las acciones semánticas.

$$A \rightarrow cd \ \{printf(c + d);\} \ B$$

3. Gramática de atributos

- La información de los atributos siempre fluye de abajo hacia arriba, de arriba a abajo o de izquierda a derecha, pero *nunca de derecha a izquierda*.
- Si todos los atributos son sintetizados, siempre será una Gramática de Atributos por la Izquierda (G.A.I.).

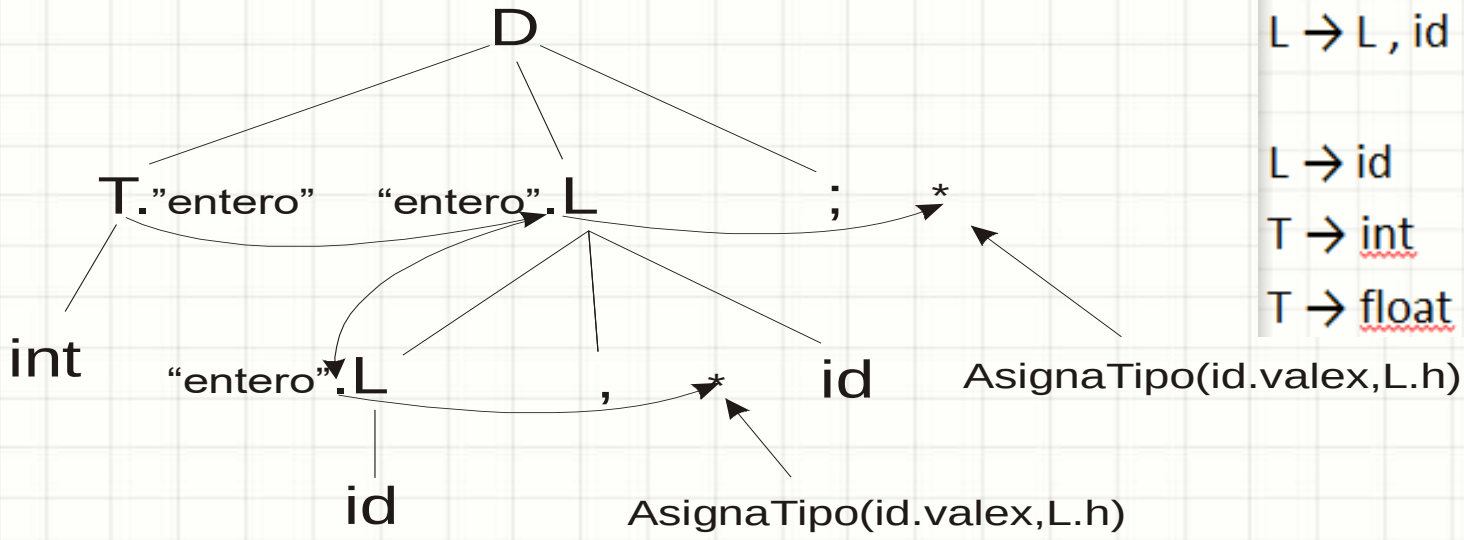




3. Gramática de atributos

Una DDS es una Gramática de atributos por la izquierda, si para cada atributo heredado $Xi.h$ que pueda aparecer, $1 \leq i \leq n$, en cualquier producción $A \rightarrow X1\ X2\ ...\ Xn$ depende sólo de:

1. Los atributos de $X1\ X2\ ...\ Xi{-}1$ (de los símbolos a su izquierda en la producción);
2. Los atributos heredados de A.



D → T L ;	{L.hered := T.tipo}
L → L , id	{L_1.hered := L_0.hered; AsignaTipo(id.valex,L_0.hered)}
L → id	{AsignaTipo(id.valex,L.hered)}
T → int	{T.tipo := “Entero”}
T → float	{T.tipo := “Real”}

4. Comprobaciones semánticas

- Un compilador debe comprobar si el programa fuente sigue tanto las convenciones sintácticas como las semánticas del lenguaje fuente, para esto se apoya de la **comprobación es**tática y la **comprobación dinámica**.

- La **comprobación estática** garantiza la detección y comunicación de algunas clases de errores de programación, en seguida se muestran ejemplos.

- La **comprobación dinámica**, se realiza durante la ejecución del programa objeto.

4. Comprobaciones semánticas

Ejemplos de comprobación estática:

- **Comprobación de tipos**: Ej., si se aplica un operador a un operando incompatible, si se suman una variable tipo matriz y una variable de función.
- **Comprobación de flujo de control**: Proposiciones que guían el flujo de control, deben tener un lugar donde transferirlo.
  - Ej., la proposición break en C hace que el control abandone la proposición que la engloba, ej. while, for o switch. Si no existiese alguna de éstas últimas, sería error.

4. Comprobaciones semánticas

Ejemplos de comprobación estática:

- **Comprobaciones de unicidad**: Ej., las variables deben declararse de forma única en una función. Otro ejemplo, las etiquetas en una proposición **case** deben ser diferentes.

- **Comprobaciones de relaciones con nombres**. En ocasiones el mismo nombre debe aparecer dos o mas veces. Ej. en lenguajes como Ada, un bloque puede tener un nombre que aparezca al principio y al final de las construcciones.

4. Comprobaciones semánticas

Ejemplos de comprobación dinámica:

- Algunas comprobaciones sólo se pueden hacer dinámicamente.

Ejemplo, si se declara:

```
int vector[10], i;
```

Y después se calcula vector[i], un compilador no puede garantizar que durante la ejecución, *i* se encuentre en el rango adecuado.

5. Comprobaciones de tipos

Con la **gestión de tipos** se asegura que el tipo de una construcción coincida con el previsto en su contexto. Ejemplos:

- Que la desreferenciación se aplique sólo a un apuntador.
- Que la indización se haga sólo sobre un arreglo
- Que en una función definida por el usuario, se aplique la cantidad y tipo correcto de argumentos, etc.

5. Comprobaciones de tipos

Un sistema de tipos seguro, elimina la necesidad de comprobar dinámicamente errores de tipos.

D → T L ; {L.hered := T.tipo}

L → L , id {L_1.hered := L_0.hered;
AsignaTipo(id.valex,L_0.hered)}

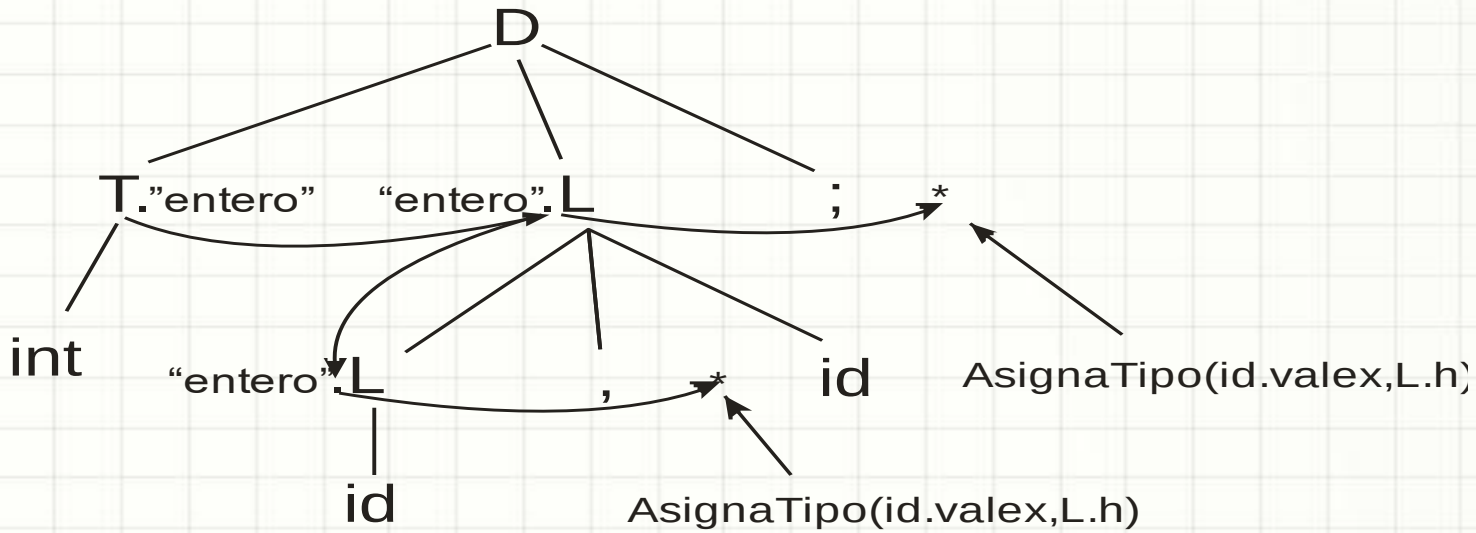
L → id {AsignaTipo(id.valex,L.hered)}

T → int {T.tipo := “Entero”}

T → float {T.tipo := “Real”}

Ejemplo de programa:

int a,b;



5. Comprobaciones de tipos

Ejemplo 2. Considerar expresiones como x+i

x es de tipo real

i es de tipo entero

Gramática:

E→E+T | T

T→num.num|num

w=3+4.5+3.8

Producciones	Reglas semánticas
E→E+T	E.tipo:= if E1.tipo=integer and T.tipo=integer integer else if E1.tipo=integer and T.tipo=real real else if E1.tipo=real and T.tipo=real real else if E1.tipo=real and T.tipo=integer real else error_tipo
E→T	E.tipo:=T.tipo
T→num.num	T.tipo:=real
T→num	T.tipo:=integer

Tabla de símbolos

- También se le llama *tabla de nombres ó tabla de identificadores* y tiene dos funciones principales:
  - Efectuar chequeos semánticos.
  - Generación de código.

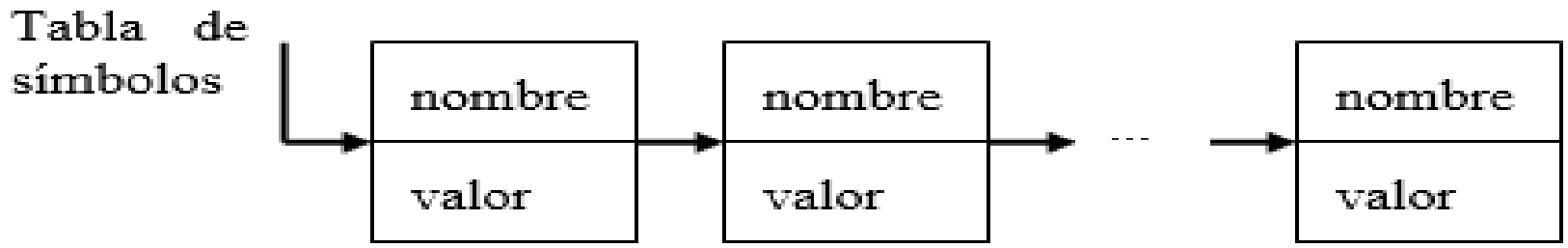
- Permanece sólo en tiempo de compilación, no de ejecución, excepto en aquellos casos en que se compila con opciones de depuración.
- En un intérprete, dado que la compilación y ejecución se producen a la vez, la tabla de símbolos permanece todo el tiempo.
- La tabla almacena la información que en cada momento se necesita sobre las variables del programa, información tal como: nombre, tipo, dirección de localización, tamaño, etc.

Tabla de símbolos

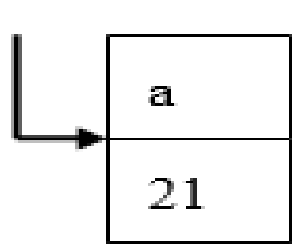
Ejemplo de la representación de una tabla de símbolos para las siguientes instrucciones:

a=7*3
b=3*a
a=a+b

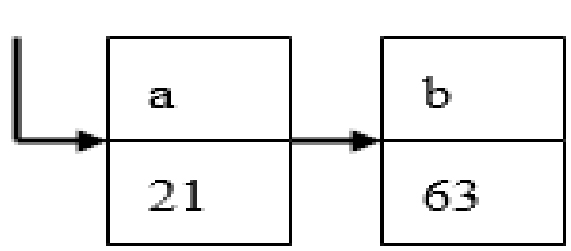
Se utilizará una lista de pares de datos



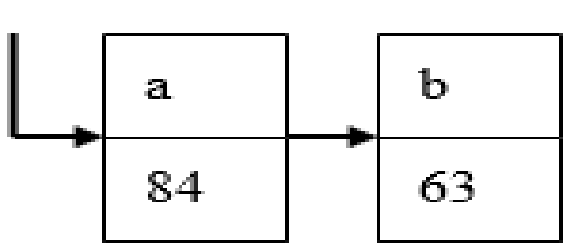
1ª operación a=7*3



2ª operación b=3*a



3ª Operación a=a+b



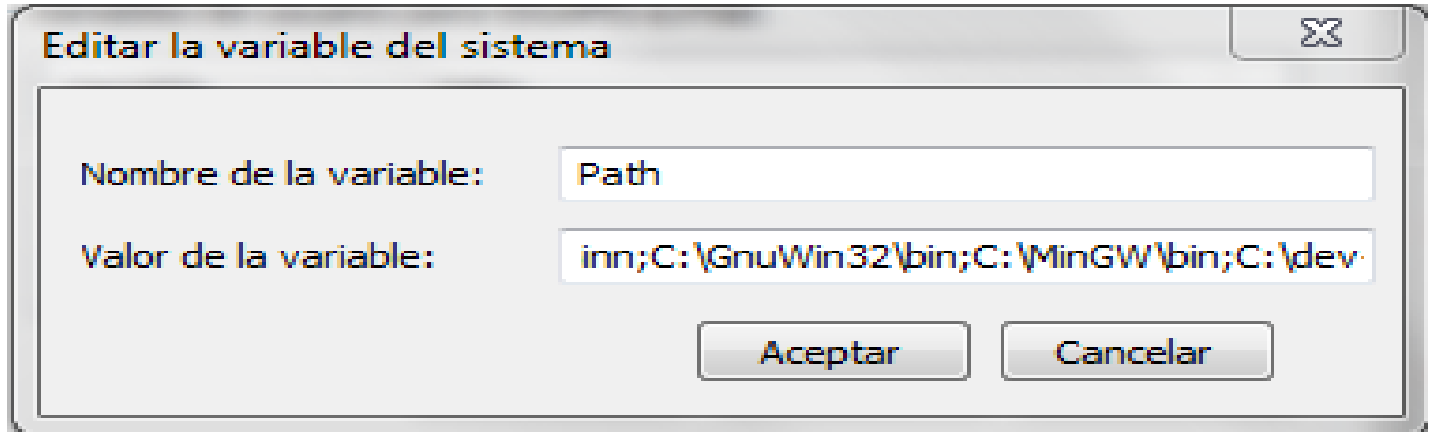
6. Generadores de analizadores léxicos.

Instalar la herramienta Flex, para el desarrollo de analizadores léxicos.

- flex-2.5.4a-1
  http://gnuwin32.sourceforge.net/packages/flex.htm
  Opción "Complete package, except sources"
- Recomendable instalar en C:\GnuWin32

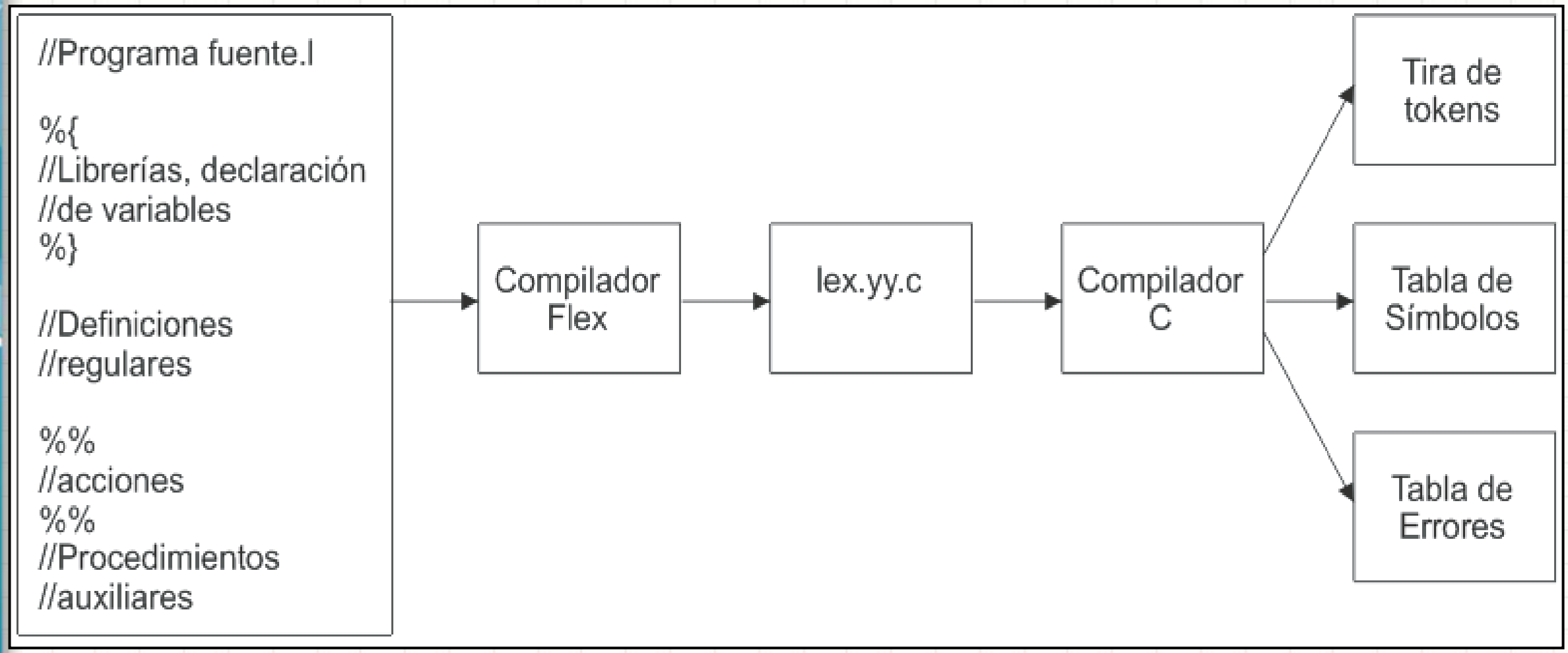
Agregar en la variable Path del sistema, la ruta donde se instaló Flex, en este caso C:\GnuWin32

Así mismo, en caso de tener instalado un compilador de C, ejemplo Dev-C, agregar también la ruta donde se encuentra instalado en la misma variable Path. De lo contrario, instalar un compilador de C.



6. Generadores de analizadores léxicos.

Flujo del analizador léxico flex.



```
/* Esto indica a Flex que lea sólo un fichero de entrada */
%option noyywrap
%{
/* Código en C que será copiado */
#include <stdio.h>
%}
//En este ejemplo no hay definición de expresiones regurales
%%
   /*** Sección de reglas y acciones ***/

   /* [0-9]+ identifica una cadena de uno o más dígitos */
[0-9]+  {
          /* yytext es una cadena que contiene el texto coincidente. */
          printf("Encontrado un entero: %s\n", yytext);
        }

.       {  /* Ignora todos los demás caracteres. */   }

%%
/*** Sección de código en C ***/
int main(void)
{
   /* Ejecuta el ''lexico'', y después termina. */
   yylex();
   return 0;
}
```

Se muestra un primer ejemplo de programa fuente (analizador léxico) en Flex.

```
%option noyywrap

%{
/*
Ejemplo de entrada al generador lex
Señala los números e identificadores
*/
#include <stdio.h>
FILE *tiraTokens=NULL, *errores=NULL;

int numLinea=1;
void escribirtiraTokenss(char token[]);
void escribirError();
%}

letra            [a-zA-Z]
digito           [0-9]
alfanumerico     [a-zA-Z0-9]
identificador    {letra}{alfanumerico}*
nint             {digito}+
nfloat           {digito}+(\.{digito}+)?(E[\+|\-]?{digito}+)+

%%
[\n]                ++numLinea;
"float"             {printf("%-15d%-20s%10s\n",numLinea, yytext,"float");
                    escribirtiraTokenss("float");
                    }
"int"               {printf("%-15d%-20s%10s\n",numLinea, yytext,"int");
                    escribirtiraTokenss("int");}

"main"              {printf("%-15d%-20s%10s\n",numLinea, yytext,"main");
                    escribirtiraTokenss("main");}
"return"            {printf("%-15d%-20s%10s\n",numLinea, yytext,"return");
                    escribirtiraTokenss("return");}
```

Se lista el programa fuente de nombre **lexico.l** en el lenguaje **Flex,** que permitirá crear el analizador léxico para las expresiones regulares que se definen en el.

```
{identificador}      {printf("%-15d%-20s%10s\n",numLinea, yytext,"id");
                     escribirtiraTokenss("id");}

{nint}               {printf("%-15d%-20s%10s\n",numLinea,yytext,"nint");
                     escribirtiraTokenss("nint");}

{nfloat}             {printf("%-15d%-20s%10s\n",numLinea,yytext,"nfloat");
                     escribirtiraTokenss("nfloat");}

[,]                  {printf("%-15d%-20s%10s\n",numLinea,yytext,",");
                     escribirtiraTokenss(",");}

[;]                  {printf("%-15d%-20s%10s\n",numLinea,yytext,";");
                     escribirtiraTokenss(";");}

[ \t\v\f]            {/*ignorar*/}

.                    {escribirError();}
%%

main(int argc, char *argv[]){
yyin = fopen("prueba1.c","r");
tiraTokens=fopen("tiraTokens.txt","w");
errores=fopen("errores.txt","w");
if(yyin!=NULL){
    printf("%-15s%-20s%10s\n","num linea","lexema","token");
    fprintf(tiraTokens,"%-15s%-20s%10s\n","num linea","lexema","token");
    fprintf(errores,"Tabla de errores lexicos\n");
    yylex();
    }
}
```

```
void escribirtiraTokenss(char token[]){
fprintf(tiraTokens,"%-15d%-20s%10s\n",numLinea, yytext,token);
}

void escribirError(){
fprintf(errores,"%-15d%s%s%s\n",numLinea,":Error lexico ",yytext," no definido");
}
```

- Instrucciones de compilación:

flex lexico.l

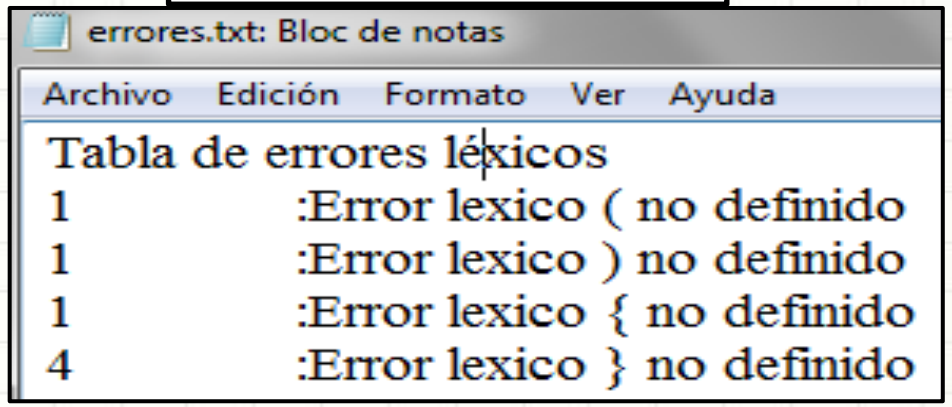
gcc –o lexico lex.yy.c

- Instrucciones de ejecución:

lexico

6. Generadores de analizadores léxicos.

Tabla de errores



Prueba1.c

```
int main(){
int a,b;
return 0;
}
```

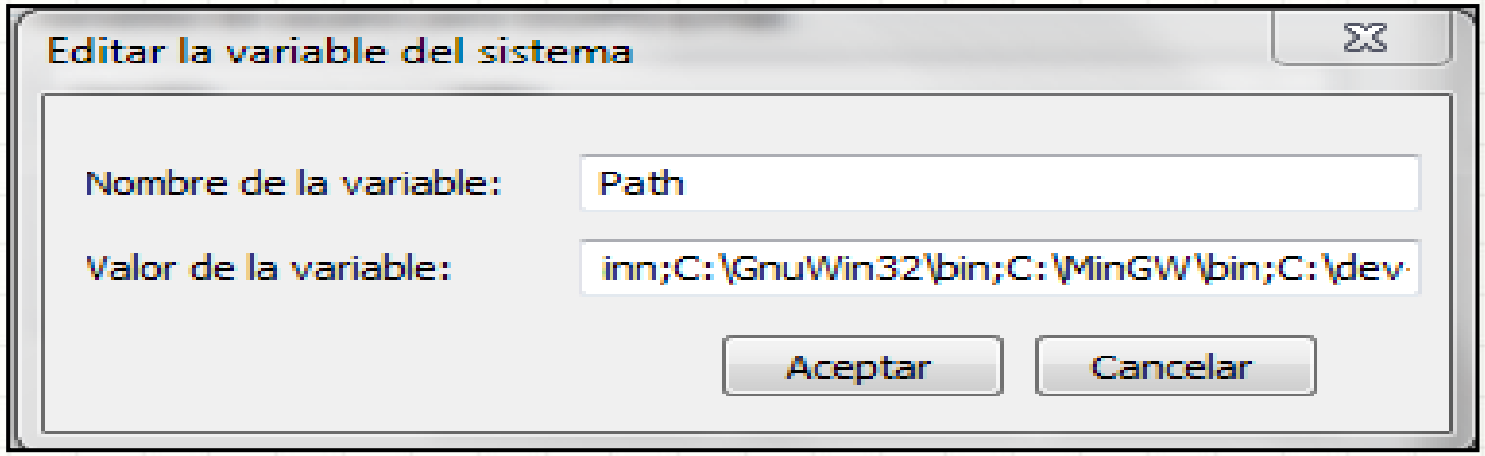
Tira de tokens

```
D:\misProgFlexBison\analizadorLexico1>lexico
num linea       lexema                  token
1               int                       int
1               main                     main
1               (                            (
1               )                            )
1               {                            {
2               int                       int
2               a                          id
2               ,                            ,
2               b                          id
2               ;                            ;
3               return                 return
3               0                         nint
3               ;                            ;
4               }                            }
```

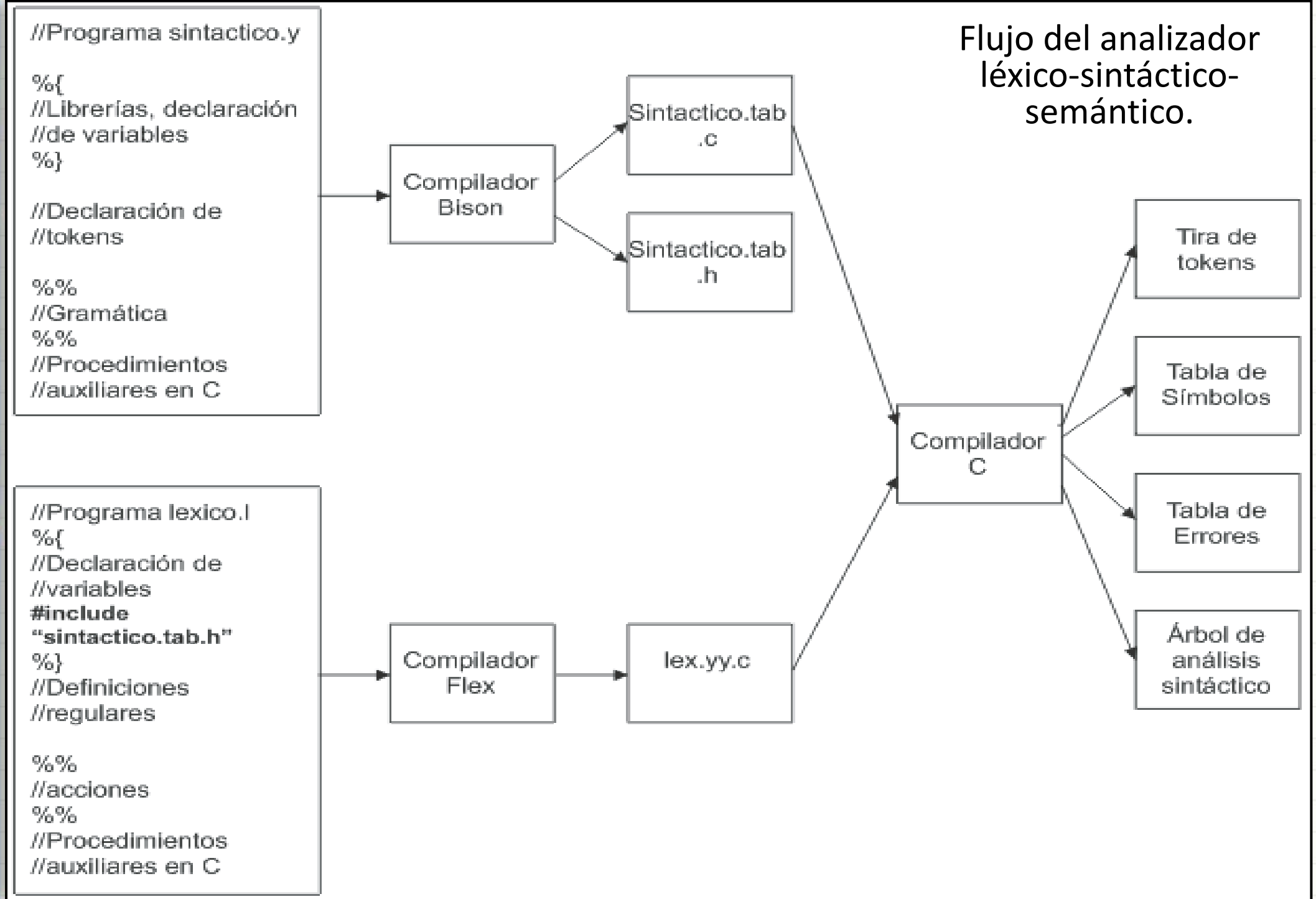
6. Generador de analizador sintáctico

Instalar Bison, generador de analizador sintáctico y semántico.

- bison-2.4.1-setup
  https://sourceforge.net/projects/gnuwin32/files/bison/2.4.1/
- Recomendable instalar en C:\GnuWin32



Flujo del analizador léxico-sintáctico-semántico.



6. Generador de analizador sintáctico

A continuación se muestra un ejemplo del analizador sintáctico para la gramática de declaración de variables arreglos en C.

$S \rightarrow TV;$

$V \rightarrow id\ V'$

$V' \rightarrow [nint]V'$

$V' \rightarrow \varepsilon$

$T \rightarrow int$

6. Generador de analizador sintáctico

Primero se lista el analizador léxico de nombre **lexico.l** para los tokens que reconoce la gramática de declaración de arreglos.

Gramática

S→TV;

V→id V'

V'→[nint]V'

V'→ε

T→int

```
%option noyywrap

%{
#include<stdio.h>
#include "sintactico.tab.h"
#define YYSTYPE char *

int numLinea=1;
%}
D    [0-9]
L    [a-zA-Z]

%%
"int"              return(INT);
{D}+               return(NINT);
";"                return(PUNTOYC);
"["                return(CORCHETEABRE);
"]"                return(CORCHETECIERRA);
{L}({L}|{D})*      return(ID);
.          /*reconocimiento de otros caracteres*/
%%
```

```
%{
#include<stdio.h>
#define YYSTYPE char *
%}

%token PUNTOYC ID CORCHETEABRE CORCHETECIERRA INT NINT

%start S /*simbolo inicial de la gramática*/

%%
/*Gramatica*/
S: T V PUNTOYC {printf("S -> T V ;\n");}
;

V: ID W {printf("V -> id W\n");}
;

W: CORCHETEABRE NINT CORCHETECIERRA W {printf("W -> [ nint ]
W\n");}
|/*epsilon*/ {printf("W->epsilon\n");}
;

T:INT {printf("T->int\n");}
;
%%
```

Ahora se lista el programa fuente del analizador sintáctico de nombre **sintactico.y ,** en el lenguaje Bison, que reconoce declaración de arreglos en C.

```
extern FILE *yyin;

yyerror(s)
char *s;
{
        fflush(stdout);
        printf("\n%s\n", s);
}

main(int argc, char *argv[]){
yyin = fopen("prueba1.c","r");
if(yyin!=NULL)
    yyparse();
else
    printf("No se pudo abrir el archivo");
}
```

- Finalmente se listan los comandos para la compilación:

  bison –d sintactico.y

  flex lexico.l

  gcc –o sintactico lex.yy.c sintactico.tab.c
- Instrucciones de ejecución:

  sintactico

6. Generador de analizador semántico

- Archivo de prueba

```
int a[10];
```

- Corrida del sintáctico, donde se muestran las derivaciones correspondiente para formar el árbol de análisis sintáctico.

```
D:\misProgFlexBison\sintacticoDeclaracionArreglos>sintactico
T->int
W->epsilon
W -> [ nint ] W
V -> id W
S -> T V ;
```

6. Generador de analizador semántico

- Para la obtención del análisis semántico, al analizador léxico se agrega código para que envíe valores léxicos requeridos al analizador sintáctico-semántico.

- Al analizador sintáctico-semántico se agregan variables(atributos) para almacenar los valores léxicos, así como atributos a los símbolos gramaticales.

En las siguientes diapositivas se listan los códigos del analizador léxico y analizador sintáctico-semántico para la gramática de declaración de arreglos.

int c[10];	var c: array[0..9] of integer;

```
S→TV;                 {S.trad:="var" ‖V.trad‖T.trad‖";"}

V→id V'               {if V'.trad!=""
                            V.temp:="array[" ‖V'.trad
                      else
                            V.temp:=V'.trad
                      V.trad:=id.valex‖":"‖V.temp
                      }
V'→[nint]V'
                      {if V1'.trad=""
                            V0'.trad:="0.." ‖nint.valex-1‖"] of"
                      else
                            V0'.trad:="0.." ‖nint.valex-1‖","‖V'1.trad
                      }
V'→ε
                      {V'.trad:=""}

T→int
                      {T.trad:="integer"}
```

```
%option noyywrap

%{
#include<stdio.h>
#include "semantico.tab.h"
#define YYSTYPE char *

int numLinea=1;
%}
D    [0-9]
L    [a-zA-Z]

%%
"int"           {return(INT);}
{D}+            {sscanf(yytext,"%d",&yylval.val);
                yylval.val=atoi(yytext);
                /*printf("%d",yylval.val);*/
                return(NINT);
                }
";"             return(PUNTOYC);
"["             return(CORCHETEABRE);
"]"             return(CORCHETECIERRA);
{L}({L}|{D})*     {strcpy(yylval.trad,yytext);
                  /*printf("%s", yylval.trad);*/
                  return(ID);}
.           {/*reconocimiento de otros caracteres*/}
%%
```

Primero se lista el analizador léxico de nombre **lexico.l** para los tokens que reconoce la gramática de declaración de arreglos.

```
%{
#include<stdio.h>
#include<string.h>

char temp[100];
%}

%union{
     int val;
     char trad[100];
     }

%token <val> NINT
%token <trad> ID
%token PUNTOYC CORCHETEABRE CORCHETECIERRA INT
%type <trad> S W V T

%start S /*simbolo inicial de la gramática*/

%%
/*Gramatica*/
S: T V PUNTOYC {
     strcpy($$,"var ");
     strcat($$,$2);
     strcat($$,$1);
     strcat($$,";");
     printf("%s",$$);
     }
;
```

Ahora se lista el programa fuente del analizador sintáctico-semántico de nombre **semantico.y ,**

en el lenguaje Bison, donde en cada regla de producción que se requiera, se le ha agregado su respectiva acción semántica.

```
V: ID W {
     if((strcmp($2,"")!=0)){
          strcpy(temp," array[");
          strcat(temp,$2);
     }
     else
          strcpy(temp,$2);
     strcpy($$,$1);
     strcat($$,":");
     strcat($$,temp);
     }
;

W: CORCHETEABRE NINT CORCHETECIERRA W {
     $2=$2-1;
     itoa($2,temp,100);
     if((strcmp($4,"")==0)){
          strcpy($$,"0..");
          strcat($$,temp);
          strcat($$,"] of");
     }
     else {
          strcpy($$,"0..");
          strcat($$,temp);
          strcat($$,",");
          strcat($$,$4);
     }
     }
     |{strcpy($$,"")};/*epsilon*/
;
```

Continuación del programa fuente **semantico.y**

```
T:INT {strcpy($$," integer");}
;
%%
extern FILE *yyin;

yyerror(s)
char *s;
{
        fflush(stdout);
        printf("\n%s\n", s);
}

main(int argc, char *argv[]){
yyin = fopen("prueba1.c","r");
if(yyin!=NULL)
    yyparse();
else
    printf("No se pudo abrir el archivo");
}
```

- Finalmente se listan los comandos para la compilación:

  bison –d semantico.y

  flex lexico.l

  gcc –o semantico lex.yy.c semantico.tab.c
- Instrucciones de ejecución:

  semantico

6. Generador de analizador semántico

- Ejemplo del archivo de prueba Prueba1.c

```
int a[5][10];
```

- Corrida del análisis semántico.

```
D:\misProgFlexBison\semanticoDeclaracionArreglos>semantico
var a: array[0..4,0..9] of integer;
```

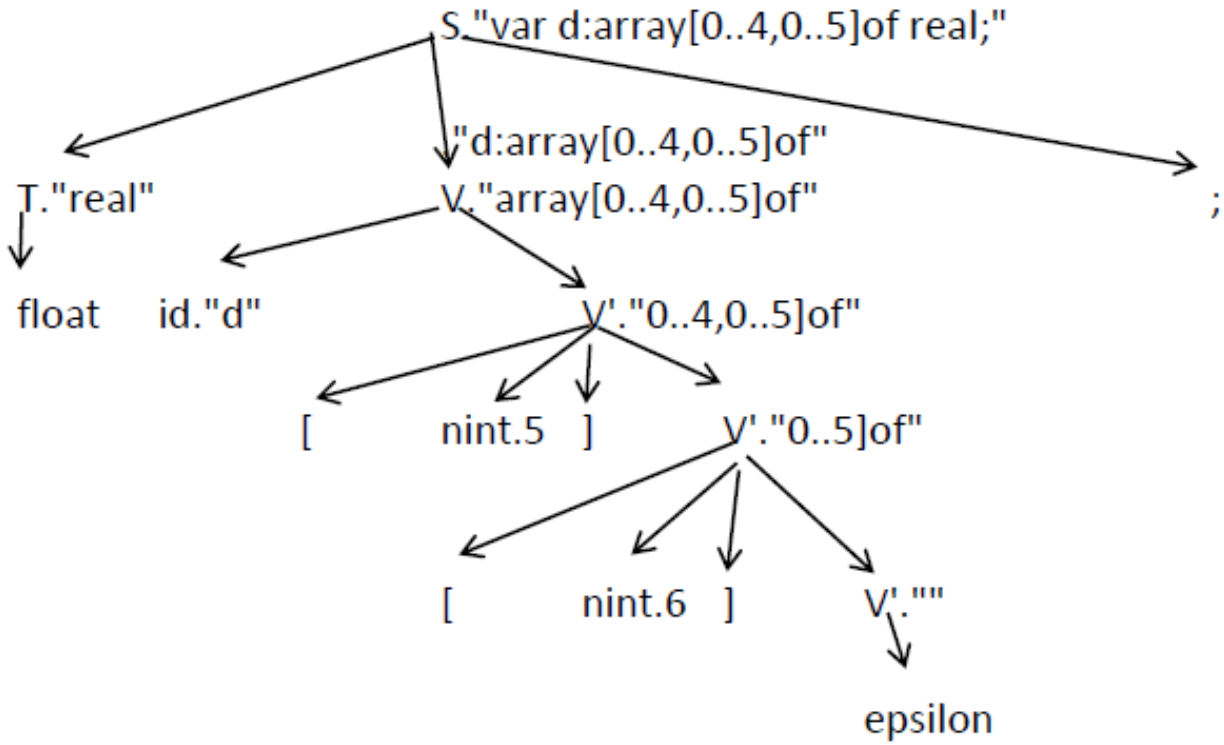
6. Generador de analizador semántico

EJEMPLO DE DEFINICION DIRIGIDA POR LA SINTAXIS

int c[10];	var c: array[0..9] of integer;
float d[5][6];	var d:array[0..4,0..5] of real;
char e;	var e:char;
char f[3][5][7];	var f:array[0..2,0..4,0..6] of char;

S→TV;	{S.trad:="var" ‖V.trad‖T.trad‖";"}
V→id V'	{if V'.trad!="" V.temp:="array["‖V'.trad else V.temp:=V'.trad V.trad:=id.valex‖":"‖V.temp }
V'→[nint]V'	{if V_1'.trad="" V_0'.trad:="0.."‖nint.valex-1‖"] of" else V_0'.trad:="0.."‖nint.valex-1‖","‖V'$_1$.trad }
V'→ε	{V'.trad:=""}
T→int	{T.trad:="integer"}
T→char	{T.trad:="char"}
T→float	{T.trad:="real"}

float id."d"[nint.5][nint.6];



Referencias

1. Compiladores. Principios, Técnicas y Herramientas (2a ed.). Aho, A.V. Pearson Educación. 2008.
2. Construcción de Compiladores. Principios y práctica. Louden, K. C. Thomson Editores. 2005