INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

COMPILADORES

“Guia”

Primer departamental

Profesor:

Ing. Tecla Parra Roberto

Grupo

3 CM8

Defina compilador
Programa que lee un programa escrito en un lenguaje fuente y lo traduce a un programa equivalente o en lenguaje objeto.

Cuáles son las dos partes de la compilación:
Análisis y síntesis

Describa las 6 fases de un compilador:
Análisis lineal o léxico, análisis sintáctico, análisis semántico, generación de código intermedio, optimización de código, generación de código.

¿Para qué sirve el Análisis Léxico?
a) Para generar el código en lenguaje objeto
b) Nos dice si una cadena pertenece al lenguaje generado por una gramática (**C**)
c) Para dividir una cadena en tokens
d) Los compiladores no lo necesitan nunca

Es una gramática que tiene cuatro componentes:
1. Un conjunto de componentes léxicos.
2. Un conjunto de no terminales.
3. Un conjunto de producciones, en el que cada producción consta de un no terminal, llamado *lado izquierdo* de la producción, una flecha y una secuencia de componentes léxicos y no terminales, o ambos, llamado *lado derecho* de la producción.

La denominación de uno de los no terminales como símbolo *inicial.*
a) Gramática Asociativa por la izquierda
b) Gramática recursiva (**C**)
c) Gramática libre de contexto
d) Gramática *ambigua*

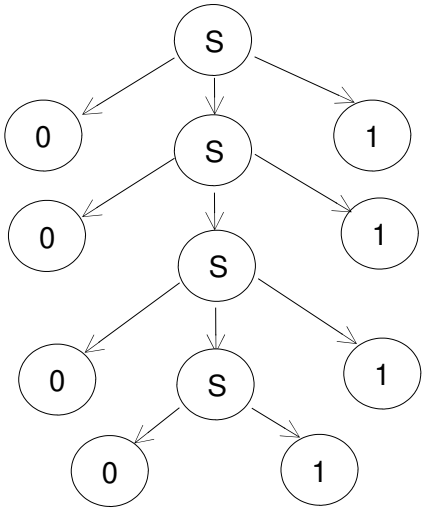
Es una gramática que puede tener más de un árbol de análisis sintáctico que genere una cadena dada de componentes léxicos.
a) Gramática Asociativa por la izquierda
b) Gramática recursiva (**D**)
c) Gramática libre de contexto
d) Gramática *ambigua*

Falso o verdadero (F/V)
Componente léxico es sinónimo de no terminal (**F**)
Análisis sintáctico ***descendente*** *es donde la construcción del árbol de análisis sintáctico se inicia en las hojas y avanza hacia la raíz (**F**)*

Considere la siguiente gramática:

*S-> **0** S **1** | **01***
a) Mostrar una derivación de ***00001111***
S -> 0 S 1
-> 0 0 S 1 1
-> 0 0 0 S 1 1 1
-> 0 0 0 0 1 1 1 1

b)Árbol de análisis sintáctico



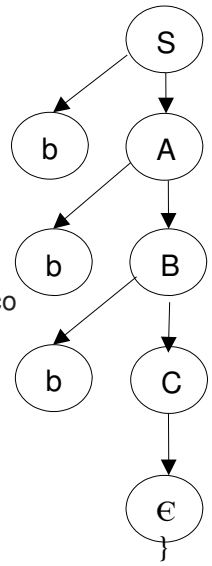
Considere la siguiente gramática

S → bA
A → bB
B → bC
C → **ε**

a) Mostrar una derivación de **bbb**

S -> b A
-> b b B
-> b b b C
-> b b b

b) Dibuje el árbol de análisis sintáctico



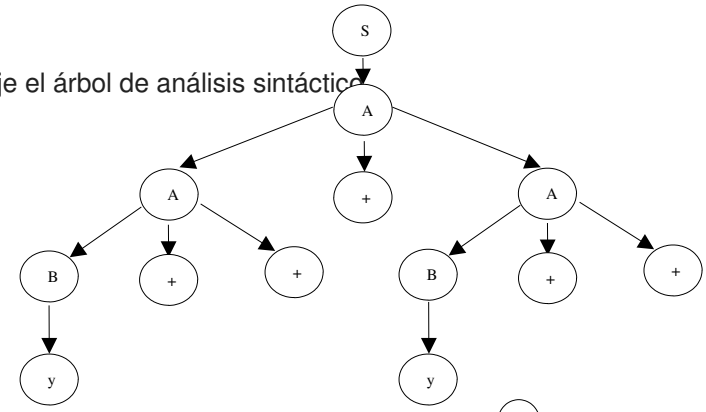
Considere la siguiente gramática

S → A
A → A+A | B++
B → y

a) Mostrar una derivación de **y + + + y + +**

S -> A
-> A + A
-> B + + + B + +
-> y + + + y + +

b) Dibuje el árbol de análisis sintáctico



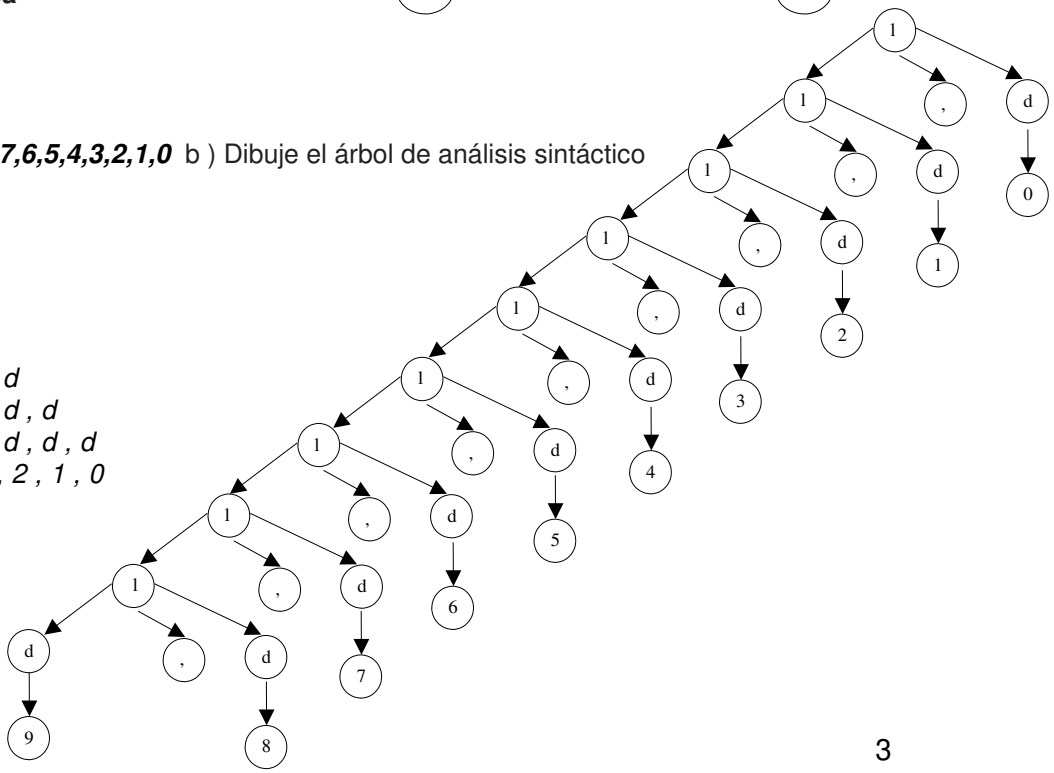
Considere la siguiente gramática

I-> I , d | d
*d-> **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |**8** | **9***

a) Mostrar una derivación de ***9,8,7,6,5,4,3,2,1,0***

I -> I , d
I -> I , d , d
I -> I , d , d , d
I -> I , d , d , d , d
I -> I , d , d , d , d , d
I -> I , d , d , d , d , d , d
I -> I , d , d , d , d , d , d , d
I -> I , d , d , d , d , d , d , d , d
I -> I , d , d , d , d , d , d , d , d , d
I -> 9 , 8 , 7 , 6 , 5 , 4 , 3 , 2 , 1 , 0

b) Dibuje el árbol de análisis sintáctico



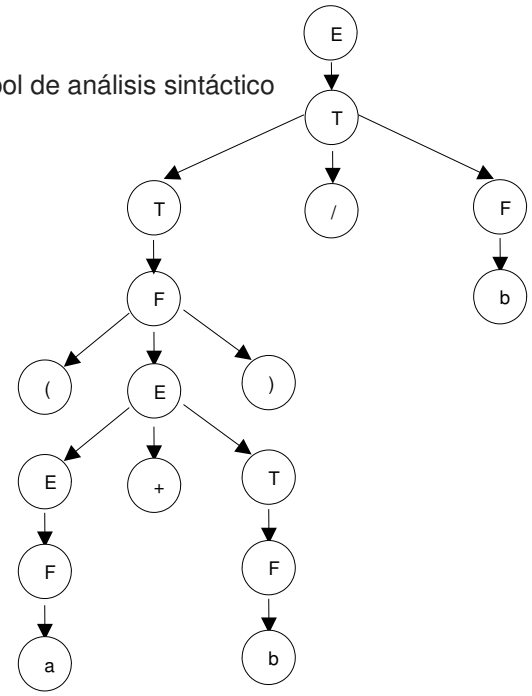
Dada la gramática

T={**a**, **b**, +, -, *, /, (,)}, N={E, T, F} S={E}
P={ E->T | E+T | E-T
  T->F | T*F | T/F
  F-> **a** | **b** | (E) } y la cadena **(a+b)/b**

a) Obtenga una derivación de dicha cadena

E -> T
-> T/F
-> F/b
-> (E)/b
-> (E + T)/b
-> (F + F)/b
-> (a + b)/b

b) Dibuje el árbol de análisis sintáctico



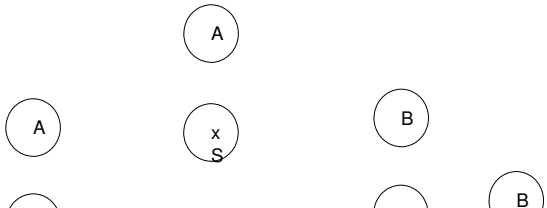
Análisis sintáctico predictivo descendente recursivo
Considere la siguiente gramática

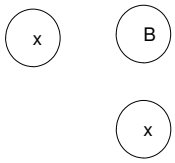
S →**a** | **(** S **)**
Escriba el analizador sintáctico predictivo descendente recursivo

```
void parea(complex +){
          if(preanalisis ==  +)
            preanalisis == sigcomplex();
          else error();
}
void  S() {
          if( preanalisis == '('){
            parea ('(');
            S (  );
            parea ( ')');
          }
          else if (preanalisis=='a')
            parea('a');
          else
            error();
}
```

AMBIGÜEDAD
Demostrar que la siguiente gramática es ambigua usando la cadena xxxxx

A →A **x** B | **x**
B → **x** B | **x**
A -> A x B

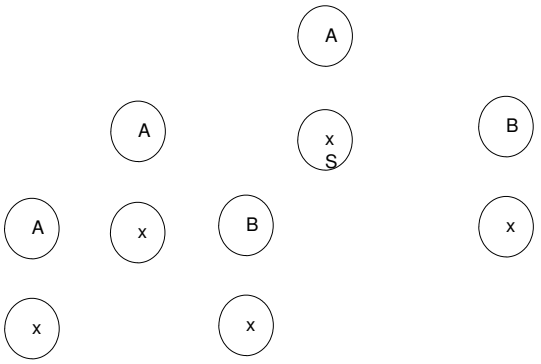




-> A x x B

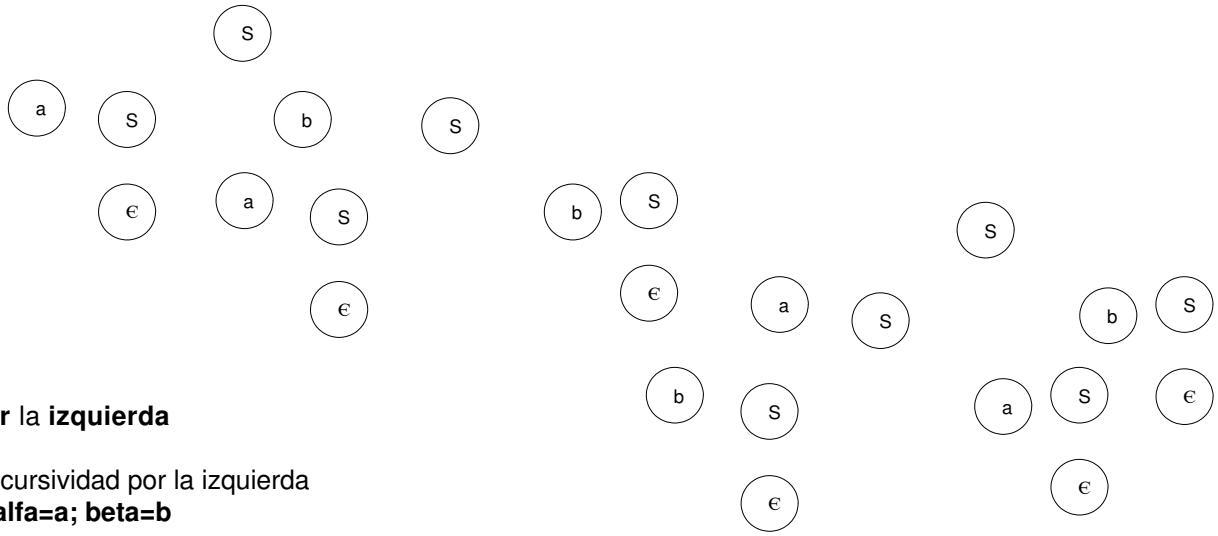
-> A x x x B
-> A x x x x
-> x x x x x

A -> A x B
-> A x B x B
-> x x x x x



Demostrar que la siguiente gramática es ambigua usando la cadena abab
S-> **a** S **b** S | **b** S **a** S | **ε**
S ->a S b S
-> a b S a S b ε
-> a b ε a ε b
-> a b a b

S -> a S b S
-> a ε b a S b S
-> a b a ε b ε
-> a b a b



Recursividad por la **izquierda**

Para eliminar la recursividad por la izquierda
A -> A**a** | **b** **alfa=a; beta=b**
se transforma en
A -> **b**R
R -> **a**R | **ε**

Ahora considere la siguiente gramática
S-> **(** L **)** | **a** //**No tiene recursividad**

Elimine la recursividad por la izquierda de dicha gramática.
L-> L **,** S | S
Beta=S
Alfa= , S

L->SR
R-> , S | **ε**

Escriba el analizador sintáctico predictivo descendente recursivo

```
void parea(complex +){
            if(preanalisis ==  +)
              preanalisis == sigcomplex();

            else error();}

void S() {
            if(parea == '(' ){
              parea ('(');
            L(  );
parea ( ')');
            }
  else if(parea == ´a´)
              parea('a');
            else
            error();}

void L( ) {
            S( );
            R( );}

void R() {
            if(parea == ' , ' ){
              parea (',');
            S(  );}
else  ;}
```

Escriba la sección de reglas de la especificación de YACC para dicha gramática

```
%left ','
%%
            S: (' L ')
             | a
            ;
            L:  S R
            ;
            R:  ',' S
            |
            ;
%%
```

Dibuje el árbol de análisis sintáctico con anotaciones para la sig cadena

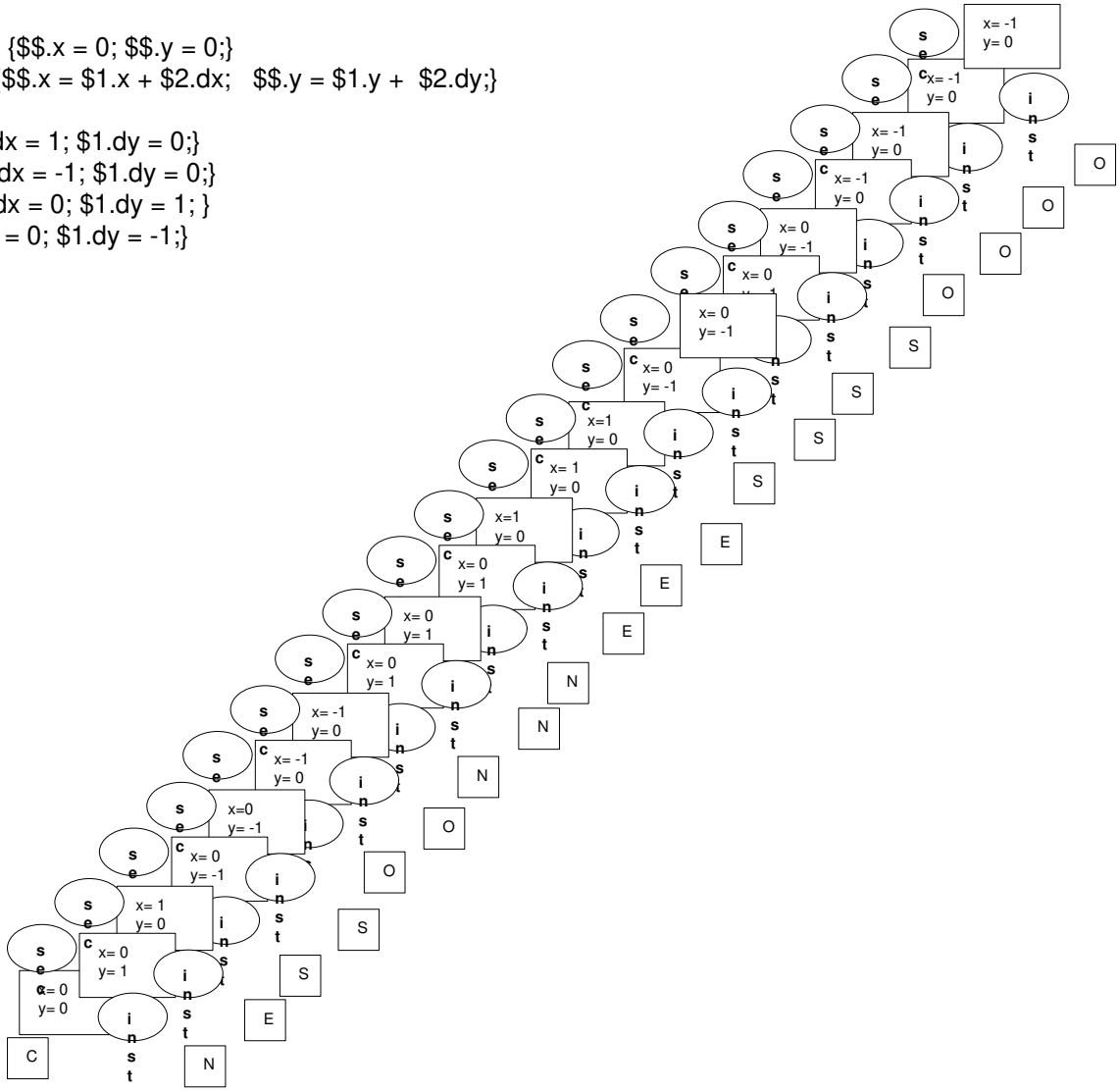
Escribir la sección de reglas de la especificación de yacc para calcular la posición final del robot.

```
%{
Struct cord{
Int x, y,dx,dy;
}
;
Typedef struct cord cordenada;
#define struct cord cordenada
#define YYSTYPE struct cord
```

```
%}

%token  comienza este oeste norte sur
%%
  sec:      comienza {$$.x = 0; $$.y = 0;}
  |         sec instr {$$.x = $1.x + $2.dx;   $$.y = $1.y +  $2.dy;}

  instr:    este {$1.dx = 1; $1.dy = 0;}
  |         oeste{$1.dx = -1; $1.dy = 0;}
  |         norte{$1.dx = 0; $1.dy = 1; }
  |         sur{$1.dx = 0; $1.dy = -1;}
  ;
%%
```



ESCRIBA UNA DEFINICIÓN DIRIGIDA POR LA SINTAXIS PARA EVALUAR EXPRESIONES BOOLEANAS.

PRODUCCIÓN	REGLA SEMÁNTICA
expr → **expr1 '|' expr2**	*expr.t = expr1.t || expr2.t*
expr → **expr1 '&' expr2**	*expr.t = expr1.t && expr2.t*
expr → **'!' expr1**	*expr.t = !expr1.t*
expr → **termino**	*expr.t = termino.t*

ESQUEMAS DE TRADUCCIÓN

Escriba un esquema de traducción para:

a) . Convertir una expresión en infijo a postfijo

```
expr -> expr + termino { print (' + ')}
expr -> expr + termino { print (' - ')}
expr -> termino
termino -> 0                    { print (' 0 ')}
termino -> 1                    { print (' 1')}
termino -> 9                { print (' 9')}
```

b) Convertir una expresión en postfijo a infijo

```
expr ->+  expr  termino +    { print (' + ');}
expr ->-  expr  termino  -   { print (' - ');}
expr -> termino
termino -> 0                    { print (' 0 ')}
termino -> 1                    { print (' 1')}
termino -> 9                { print (' 9')}
```

c) Convertir una expresión en infijo a prefijo}

```
expr -> expr  termino +    { print ('+'expr , termino}
expr -> expr  termino  -   { print ('-' exp, termino) }
expr -> termino
termino -> 0                    { print (' 0 ')}
termino -> 1                    { print (' 1')}
termino -> 9                { print (' 9')}
```

d) Convertir una expresión en prefijo a infijo

```
expr -> + expr  termino     { print (expr , '+', termino}
expr -> - expr  termino     { print (exp, '-', termino) }
expr -> termino
termino -> 0                    { print (' 0 ')}
```

termino -> 1 { print (‘ 1’)}
termino -> 9 { print (‘ 9’)}

Escritura de Gramaticas

26.- Escribir una gramática que genere todas las cadenas de longitud 4 formadas con los símbolos del alfabeto {a,b,c}

T = {a,b,c}
N = {A,S}
S = {S}
P = { S-> AAAA
A-> a|b|c}

27.- Escribir una gramática que sirva para generar las siguientes cadenas

Especie perro	Especie gato	Especie perro	Especie gato
Edad 1	Edad 2	Edad 2	Edad 2
Sexo macho	Sexo macho	Sexo hembra	Sexo macho
Tamaño grande	Tamaño mediano	Tamaño pequeño	Tamaño grande
Colores negro , blanco	Colores negro , blanco , café	Colores canela , gris	Colores blanco
Soy rápido , activo, alegre	Soy tranquilo , sociable	Soy fuerte , alegre, activo.	Soy listo , obediente
Aficiones correr, comer	Aficiones dormir, parrandear, comer	Aficiones aullar	Aficiones jugar, haraganear

S-> especie + edad + sexo + tamaño + colores +soy + aficiones
Especie-> perro|gato
Edad -> 1|2
Sexo -> macho|hembra
Tamaño -> grande|mediano|pequeño
Colores -> colores,colores| colores|negro|blanco|café|canela|gris
Soy -> soy,soy|soy|rápido|activo|alegre|tranquilo|sociable|fuerte|listo|obediente
Aficiones ->aficiones,aficiones|aficiones|correr|comer|dormir|parrandear|aullar|jugar|haraganear

28.- Escribir una gramática que sirva para generar las siguientes cadenas

Etiquetado Nerd	Etiquetado Geek	Etiquetado Nerd	Etiquetado Freak
Nivel Junior	Nivel Senior	Nivel Junior	Nivel Senior
Sexo Hombre	Sexo Mujer	Sexo Mujer	Sexo Hombre
Lenguajes Java , C , Logo	Lenguajes Pascal , Prolog , SQL	Lenguajes PHP , Perl, Java	Lenguajes Ensamblador, C
Aficiones programar, videogames, comics, hackear, googlear	Aficiones chatear, videogames, programar	Aficiones hackear, googlear, gotcha, dormir	Aficiones gotcha, dormir, chatear, comics

S-> etiquetado + nivel + sexo + lenguajes + aficiones
Etiquetado -> nerd|geek|freak
Nivel -> junior|senior
Sexo -> hombre|mujer
Lenguajes -> lenguajes,lenguajes|lenguajes|java|c|logo|pascal|prolog|php|pearl|ensamblador

Aficiones->Aficiones,aficiones|aficiones|programar|videogames|comics|hackear|googlear|chatear|dormir|gotcha

YACC

1.- Los %% se usan para indicar (b)

a)inicio de la sección de declaraciones b)inicio de la sección de reglas
c)precedencia de los operadores d)fin del código de soporte

2.- %token sirve para indicar (d)

a)inicio de la sección de declaraciones b)los no terminales de la gramática
c)precedencia de los operadores d)los terminales de la gramática

3.- Como le indica el analizador léxico (yylex) al analizador sintáctico (yyparse) que ya no hay mas tokens en la entrada (d)

a)retornando cero b)retornando -1
c)almacenando -1 en yylval d)almacenando 0 en yylval

4.- Una acción gramatical debe ir entre (d)

a) comillas b) paréntesis c) corchetes d) llaves

5.- Considere la producción

S : S ' a' S 'b'

$4 a cual de los miembros del lado derecho de la producción se refiere?

b)la 1er S

b) la b

-

Considere la siguiente gramática (los terminales se indican en negritas)

L-> L**,** D | D
D-> **0** | **1**
Escriba la sección de reglas de la especificación de yacc para dicha gramática

```
%%
        L:      L ',' D
  |       D
  ;
  D:    0
  |     1
  ;

%%
```

35.- Escriba la especificación de yacc para la gramática

S → U | V
U → T**a**U | T**a**T
V → T**b**V | T**b**T
T → **a**T**b**T | **b**T**a**T | **ε**

```
%%
  S:    U
  |     V
```

```
  ;
  U:      T ‘a’ U
  |       T ‘a’ T
  ;
  V:      T ‘b’ V
  |       T ‘b’ T
  ;
  T       /*nada*/
          | ‘a’ T ‘b’ T
  |       ‘b’ T ‘a’ T
  ;
%%
```

36.- Escriba las acciones gramaticales para que imprima el numero de b's en la cadena de entrada

```
%{
        int numb;
#define YYSTYPE
%}

%%
S : ’(’ B ’)’      {$$ = $2;}
;
B : ’(’ B ’)’      { $$ = $2;}
  | D              { $$=$1; }
;
D : { }
  | ’b’ D         { $$.numb++; $$ = $2; }
;
%%
```

37.- Considere la siguiente gramática (los terminales se indican en negritas)

lista->lista **,** figura | figura
figura-> triangulo | cuadrilatero
triangulo-> **lado lado lado**
cuadrilatero-> **lado lado lado lado**

Escriba la sección de reglas de la especificación de yacc para dicha gramática y las acciones semánticas respectivas para que se imprima si un triangulo es equilátero y si un cuadrilátero es un cuadrado.

```
%%
  lista:    lista ‘,’ figura
  |         figura
  ;
  figura:   triangulo
  |         cuadrilátero
  ;
  triangulo:        lado lado lado {if($1==$2 && $2==$3) printf(“Equilatero”);}
  ;
  cuadrilátero:     lado lado lado lado {if($1 == $2 && $2 == $3 && $3 == $4) printf(“Cuadrialtero”);}
  ;
%%
```

Considere la gramática para generar paréntesis anidados.

1.- **A → (A)**	2.- **A → a**

Construya la tabla de Análisis Sintáctico Predictivo no Recursivo.

	(	)	a	$
A	**A → (A)**		**A → a**	

Use ambos análisis para analizar las siguientes cadenas:

1.- **(a)**

Análisis Sintáctico Peredictivo no Recursivo.

Pila	Entrada	Acción
$A	(a)$	**A → (A)**
$)A(	(a)$	
$)A	a)$	**A → a**
$)a	a)$	
$)	)$	
$	$	

2.- **((a))**

Análisis Sintáctico Peredictivo no Recursivo.

Pila	Entrada	Acción
$A	((a))$	**A → (A)**
$)A(	((a))$	
$)A	(a))$	**A → (A)**
$))A(	(a))$	
$))A	a))$	**A → a**
$))a	a))$	
$))	))$	
$)	)$	
$	$	

3.- **(((a)))**

Análisis Sintáctico Peredictivo no Recursivo.

Pila	Entrada	Acción
$A	(((a)))$	**A → (A)**
$)A(	(((a)))$	
$)A	((a)))$	**A → (A)**
$))A(	((a)))$	
$))A	(a)))$	**A → (A)**
$)))A(	(a)))$	
$)))A	a)))$	**A → a**
$)))a	a)))$	
$)))	)))$	
$))	))$	

$)	)$	
$	$	

Considere la siguiente gramática:

1.- **S → a**	2.- **S → (S R**	3.- **R → , S R**	4.- **R →)**

Construya la tabla de Análisis Sintáctico Predictivo no Recursivo.

	a	(	,	)	$
S	**S → a**	**S → (S R**			
R			**R → , S R**	**R →)**	

1.- **(a)**

Análisis Sintáctico Peredictivo no Recursivo.

Pila	Entrada	Acción
$S	(a)$	**S → (S R**
$RS(	(a)$	
$RS	a)$	**S → a**
$Ra	a)$	
$R	)$	**R →)**
$)	)$	
$	$	

2.- **(a , a)**

Análisis Sintáctico Peredictivo no Recursivo.

Pila	Entrada	Acción
$S	(a,a)$	**S → (S R**
$RS(	(a,a)$	
$RS	a,a)$	**S → a**
$Ra	a,a)$	
$R	,a)$	**R → , S R**
$RS,	,a)$	
$RS	a)$	**S → a**
$Ra	a)$	
$R	)$	**R →)**
$)	)$	
$	$	

3.- **(a , a , a)**

Análisis Sintáctico Peredictivo no Recursivo.

Pila	Entrada	Acción
$S	(a,a,a)$	**S → (S R**
$RS(	(a,a,a)$	
$RS	a,a,a)$	**S → a**
$Ra	a,a,a)$	
$R	,a,a)$	**R → , S R**
$RS,	,a,a)$	
$RS	a,a)$	**S → a**
$Ra	a,a)$	
$R	,a)$	**R → , S R**
$RS,	,a)$	
$RS	a)$	**S → a**
$Ra	a)$	
$R	)$	**R →)**
$)	)$	
$	$	

4.- **(a , a , a , a)**

Análisis Sintáctico Peredictivo no Recursivo.

Pila	Entrada	Acción
$S	(a,a,a,a)$	**S → (S R**
$RS(	(a,a,a,a)$	
$RS	a,a,a,a)$	**S → a**
$Ra	a,a,a,a)$	
$R	,a,a,a)$	**R → , S R**
$RS,	,a,a,a)$	
$RS	a,a,a)$	**S → a**
$Ra	a,a,a)$	
$R	,a,a)$	**R → , S R**
$RS,	,a,a)$	
$RS	a,a)$	**S → a**
$Ra	a,a)$	
$R	,a)$	**R → , S R**
$RS,	,a)$	
$RS	a)$	**S → a**
$Ra	a)$	
$R	)$	**R →)**
$)	)$	
$	$	