

Compiladores Primer Parcial

Alfabeto: Conjunto finito, no vacío de símbolos.

Lenguajes formales: No tienen ambigüedad, pueden ser de alto o bajo nivel.

Palabra: Secuencia de símbolos sobre un alfabeto.

Símbolo: Carácter normalmente con significado.

Lenguaje: Conjunto de palabras.

Gramática: Reglas con las que se van a producir las palabras (por extensión).

Conjunto de no terminales o ramas (N): Un conjunto de símbolos (normalmente se representan con mayúsculas) sirven para producir otros símbolos.

Conjunto de terminales (T): Un conjunto de símbolos (normalmente con minúsculas) estos ya no producen ningún símbolo, por lo cual se denominan hojas.

Producciones (P): Conjunto de reglas que indican como se transformas los NO terminales.

$$S \rightarrow AB$$
$$A \rightarrow a \mid aA \mid \lambda$$
$$B \rightarrow b \mid bB \mid \lambda$$

Raíz: $R \in N$ y es el símbolo inicial.

Dada una cadena v perteneciente a un alfabeto A , se tiene que:

$$\sum_i^{|v|} |v|_i = |v|$$

Longitud absoluta: es la longitud de una cadena, es la cantidad de símbolos totales que hay en una cadena.

Longitud relativa: de una palabra $|casa|_a=2$

Un conjunto se puede definir de 2 formas:

♥ **Extensión:** se enumeran los elementos.

$A = \{a, e, i, o, u\}$

$B = \{x / 0 < x < 10, \text{ es primo}\}$

♥ **Comprensión:** las características que definen los elementos.

$A = \{\text{vocales}\}$

$B = \{2, 3, 5, 7\}$

Si el conjunto es **infinito** solo puede definirse por **comprensión**.

Las gramáticas y un lenguaje son equivalentes.

Expresión regular	Lenguaje
ab	ab
$[ab]$	a, b
a^+b	$ab, aab, aaab, aaaaab \dots$
a^+b^+	$ab, aab, aaab, abb, aabb \dots$
$[ab]^+$	$a, b, aa, ab, ba, bb, aaa, abb, aba \dots$
$[ab][12]$	$a1, a2, b1, b2$

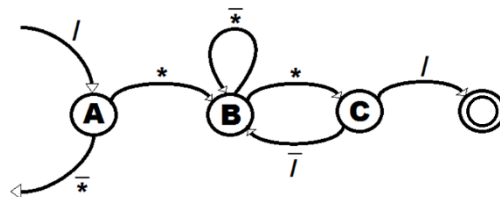
Gramática regular \leftrightarrow Lenguaje regular \leftrightarrow Expresión regular \leftrightarrow Autómata finito

Gramática para comentarios:

`/* [((abc..z 01233..9!"·$%)] ^*-* /)*/`

`/* jkhghighg */`

`/* este es un *comentario* */`



Etapas de un compilador



Análisis léxico:

- ♥ Se incluyen archivos de cabecera
- ♥ Se quitan comentarios
- ♥ Se separa el programa en tokens
- ♥ Se etiquetan los tokens

Token: ficha, es una unidad léxica.

```
main(){/*hola*/  
int a, b;  
a=38;  
b=6;  
a = a + b;  
}
```

Tokens

```
main  
(  
)  
{  
Int  
a  
,  
b  
;  
A  
=  
38  
;  
B  
=  
6  
;  
a
```

Consideremos dos conjuntos:

Conjunto de separadores:

- ♥ Espacio
- ♥ Tabulador
- ♥ Salto de línea
- ♥ Fin de archivo

No son tokens:

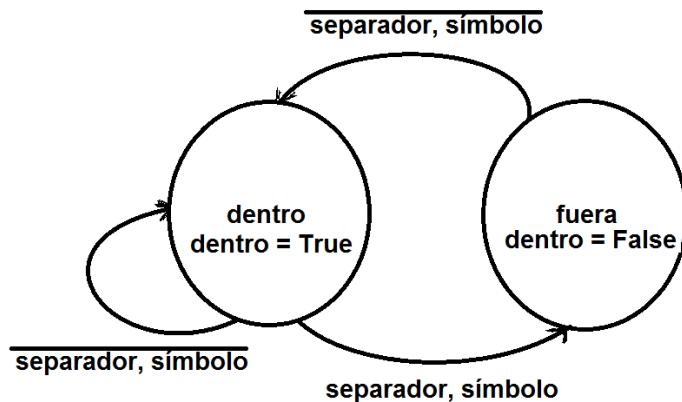
Conjunto de símbolos especiales:

`!#$%&/*+-.=:;[]{}`

`<=` Símbolo compuesto

Son tokens:

Todo lo que no sea símbolo especial ni separador es un token y está delimitado por un símbolo especial o un separador.



Etiquetar tokens:

`main` Palabra reservada

`(` Símbolo

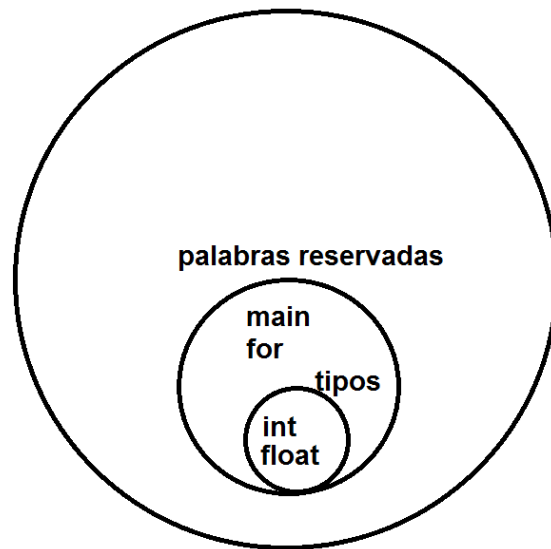
`)` Símbolo

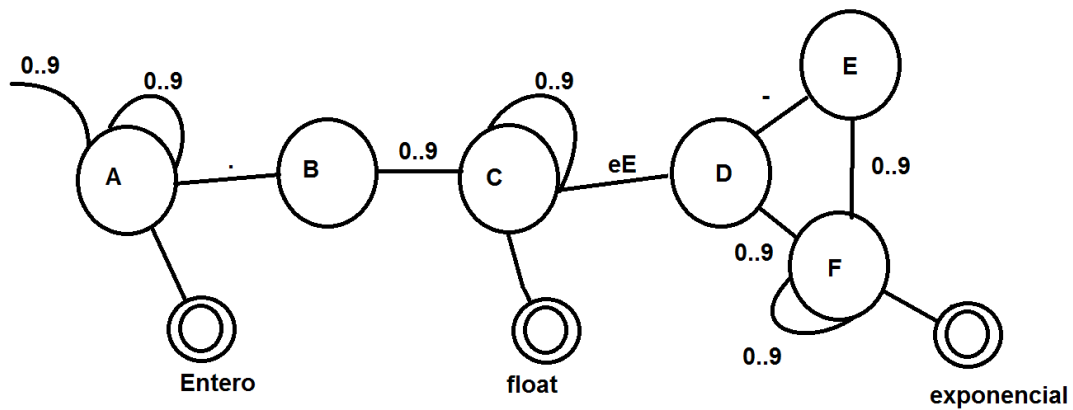
`{` Símbolo

`int` Palabra reservada (tipo de dato)

`var1` identificador

, Símbolo
var2 identificador
; Símbolo
var1 identificador
= Símbolo
38 Número entero
; Símbolo
var2 identificador
= Símbolo
6 Número entero
; Símbolo
var1 identificador
= Símbolo
var1 identificador
+ Símbolo (operador)
var2 identificador
; Símbolo
} Símbolo



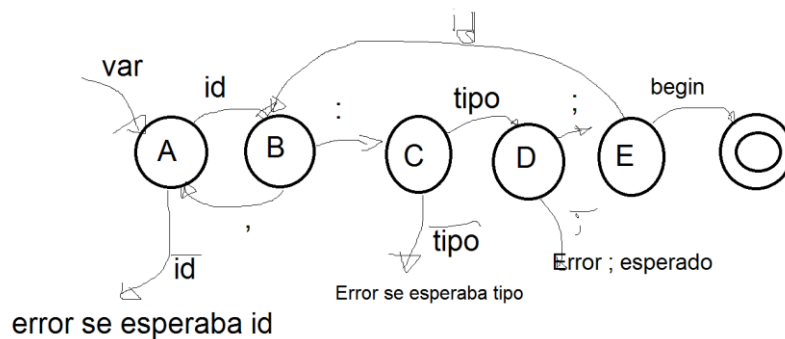


Análisis sintáctico:

Verifica que la estructura de las frases sea correcta.

- ♥ El perro corre
- ♥ La perro corre
- ♥ El perros corre
- ♥ El la los las corre

Ejemplo en pascal:



Expresiones:

Expresión: $2 + 2 > 4$ Evaluación

Expresión: $5 > 5$ Evaluación

Arboles sintácticos:

Existen tres formas de recorrer un árbol:

Pre orden (notación prefija): primero leemos la raíz, después el nodo izquierdo y por último el nodo derecho.

raiz- izq - der

In orden (notación infija): primero leemos el nodo izquierdo, después la raíz y por último el nodo derecho.

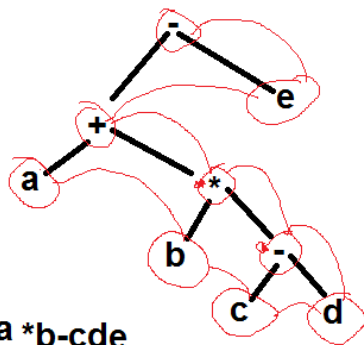
izq - raiz - der

Pos orden (notación posfija): primero nodo izquierdo, después nodo derecho y por último nodo raíz.

izq - der- raíz

Los árboles se leen de abajo para arriba.

a+b*(c-d)-e



prefija -+a *b-cde

posfija abcd-*+e-

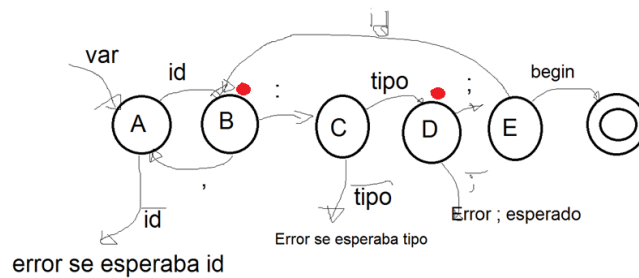
Análisis semántico:

Verificación de los tipos de datos.

Ejemplo: no hay errores sintácticos, pero no tiene sentido lo que hace.

```
Var
a : integer;
a : string;
begin
  kjkjk

  var
  var1, var2 : real;
  var3 : real;
  begin
    var2 =
    (var1*var3)/var2;
```



¿En qué etapa del compilador se realiza la verificación de tipos? En la etapa del análisis semántico.

Tabla de variables:

Nombre variable	Tipo
Var1	real
Var2	real
Var3	real

Operaciones válidas

Flotante + flotante -> flotante

Flotante - flotante -> flotante

Flotante * flotante -> flotante

Flotante / flotante -> flotante

Entero + entero -> entero

Entero - entero -> entero

Entero * entero -> entero

Entero / entero -> flotante

Entero + flotante -> flotante

Flotante + entero -> flotante

Entero operador flotante -> flotante

Cadena + cadena -> cadena

Cadena * entero -> cadena

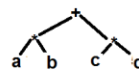
Generación de código intermedio:

A partir de los árboles sintácticos comienza la generación del código intermedio -> código maquina

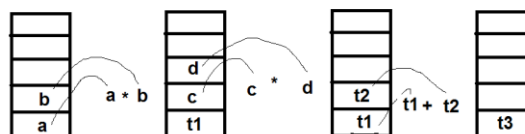
Código intermedio

```
t1 = a*b;  
t2 = c+d;  
t3 = t1+t2;
```

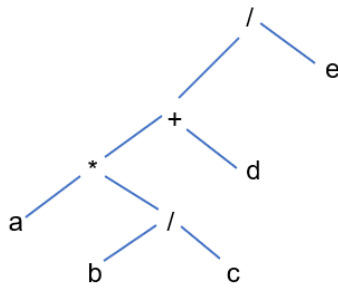
$a * b + c * d$



$ab*cd*+$

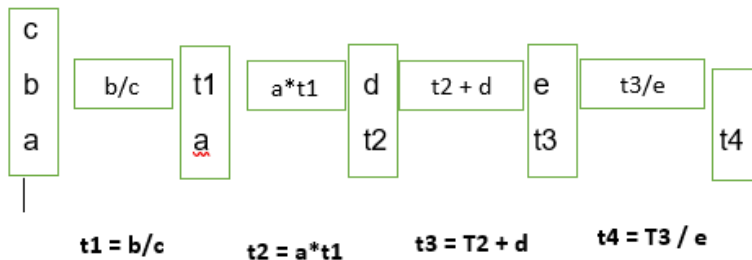


Ejemplo: $(a*(b/c)+d)/e$



Posfija: $a\ b\ c\ /\ *\ d\ +\ e\ /\$

Código intermedio:



Código ensamblador:

El código ensamblador depende de cada procesador, actualmente existen dos filosofías:

RISC (Reduced Instruction Set Computing): Pocas instrucciones, muy generales.

CISC (Complex Instruction Set Computing): Muchas instrucciones, pueden llegar a ser bastante complejas.

RISC	CISC
Unas cuantas instrucciones simples	Muchas instrucciones complejas
Instrucciones de longitud fija	Instrucciones de longitud variable
Complejidad en el compilador	Complejidad en el Microcodigo
Acceso a la memoria solo con instrucciones load/store	Muchas instrucciones pueden accesar la memoria
Muy pocos modos de Direccionamiento	Muchos modos de Direccionamiento

ADD a,b,c ; Toma los valores de los registros a y b, los suma y el resultado lo guarda en el registro c

Ensamblador de una dirección (siempre vamos a tener un registro, que será el acumulador):

LDA i; carga en el acumulador el contenido de la localidad i

ADD i; suma el acumulador con el contenido de i y el resultado lo almacena en i

MUL i; multiplica el acumulador con el contenido de i y el resultado lo almacena en i

STA i; toma el valor del acumulador y lo pasa a la localidad i

Acumulador: Donde se van haciendo las operaciones.

PC: El contador

Ejemplo:

$T1 = a * b;$

$T2 = c * d;$

$T3 = t1 + t2;$

Generación de código:

LDA a;

MUL b;

STA t1;

LDA c;

MUL d;

STA T2;

LDA t1;

ADD t2;

STA t3;

0	LDA a;
1	MUL b;
2	STA t1;
3	LDA c;
4	MUL d;
5	STA T2;
6	LDA t1;
7	ADD t2;
8	STA t3;
100	2 a
101	3 b
102	4 c
103	5 d
104	6 t1
105	20 t2
106	26 t3

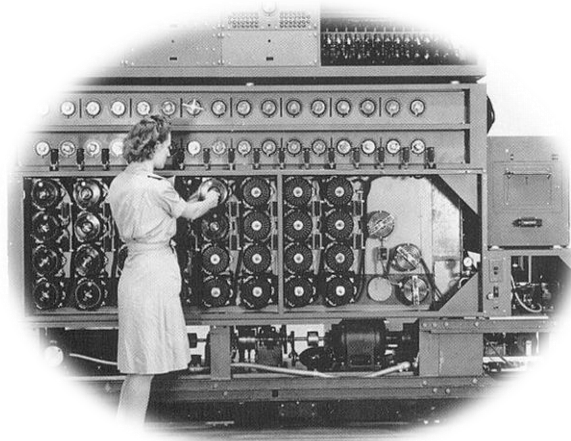


Acumulador

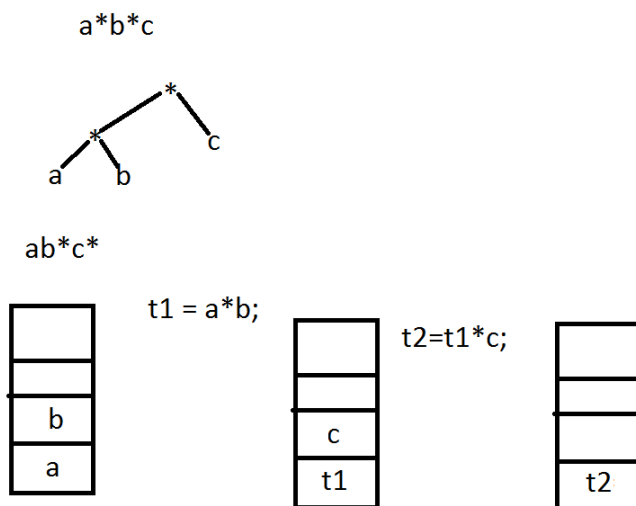
26

PC (Program counter)

La memoria en este ejemplo se puede visualizar como una cinta, que tiene un cabezal que se mueve a la derecha -> Una máquina de Turing



Nemónico: memoria, palabras muy cortas que representan lo que se va a hacer.




$t1 = a*b;$
 $t2=t1*c;$

LDA a;
 MUL b;
 STA t1;
 LDA t1;
 MUL c;
 STA t2;

Optimización:

0	LDA a;
1	MUL b;
2	STA t1;
3	LDA t1;
4	MUL c;
5	STA t2;
6	
7	
8	




100	2	a
101	3	b
102	4	c
103		
104	6	t1
105	24	t2

Acumulador

24

PC (Program counter)

0	LDA a;	
1	MUL b;	
2	MUL c;	
3	STA t2;	
4		
5		
6		

100	2	a
101	3	b
102	4	c
103		
104		
105	24	t2
106		

Acumulador

24

PC (Program counter)

LDA a;
MUL b;
STA t1;
LDA t1;
MUL c;
STA t2;

LDA a;
MUL b;
MUL c;
STA t2;

c = c + 1;

LDA c;

ADD 1;

STA c;

c++;

INC c;

La instrucción INC incrementa en 1 el valor de a, es más eficiente porque lo hace en solo un ciclo de reloj.

for (c=0; c<10; c++)

1100				
12	12	12	32	32
x2	x8	x16	x10	x100
<hr/>	<hr/>	<hr/>	<hr/>	
24	96	72	00	
		12	32	3200
11000	1100000	<hr/>	<hr/>	
		192	320	

SHL desplazamiento a la izquierda

LDA memoria -> acumulador

STA acumulador -> memoria

ADD suma

MUL multiplicación

SUB resta

DIV división

Para poder implementar ciclos se requieren saltos:

JMP salto incondicional

JZ Salto condicional, salta si es cero

a=1

b=5;

d = 2;

for (c=0; c<b; c++){

 a = a * d + d;

LDV carga en el acumulador un valor

LDV 1;

STA a; a = 1

LDV 5;

STA b; b =5;

LDV 2;

STA d; d = 2;

LDV 0;

STA c; c = 0;

1 LDA c; c<b?

SUB b;

JZ #2

LDA a;

MUL d; a = a * d + d;

ADD d;

STA a;

INC c;

JMP #1

#2

a=1

d = 2;

a = a * d + d;

a = a * d + d;

a = a * d + d;

a = a * d + d;

a = a * d + d;

Máquina virtual:

Es un programa que simula el comportamiento de una computadora.