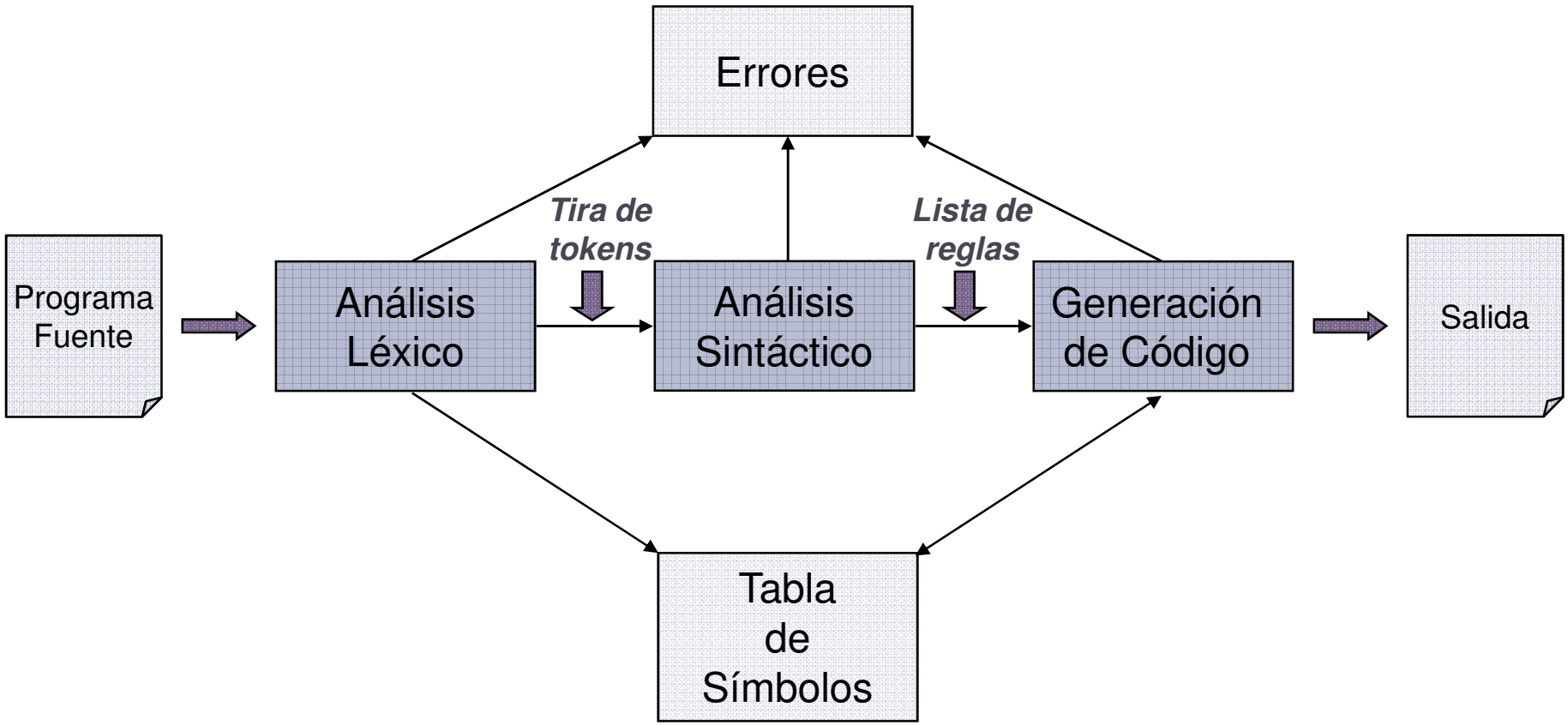
Diseño de Compiladores I

Análisis Sintáctico
Introducción a Yacc

Fases de la Compilación



Análisis Sintáctico: Estrategias de Parsing

- Agrupa los tokens del programa fuente en frases gramaticales que el compilador usará en las siguientes etapas.
- Obtiene una cadena de tokens del Analizador Léxico, y verifica que la cadena de tokens pueda generarse mediante la gramática del lenguaje.

  - Parsing Ascendente (botton up)
    - Construye el árbol desde las hojas a la raíz

  - Parsing Descendente (top down)
    - Construye el árbol desde la raíz a las hojas

Parsing Ascendente
Gramáticas LR

- Va del programa a la hipótesis.
- El programa se lee de izquierda a derecha (**L**).
- Las reglas se leen de derecha a izquierda (**R**): el lado derecho se reemplaza por el izquierdo.
- Estrategia: ***Reducción***

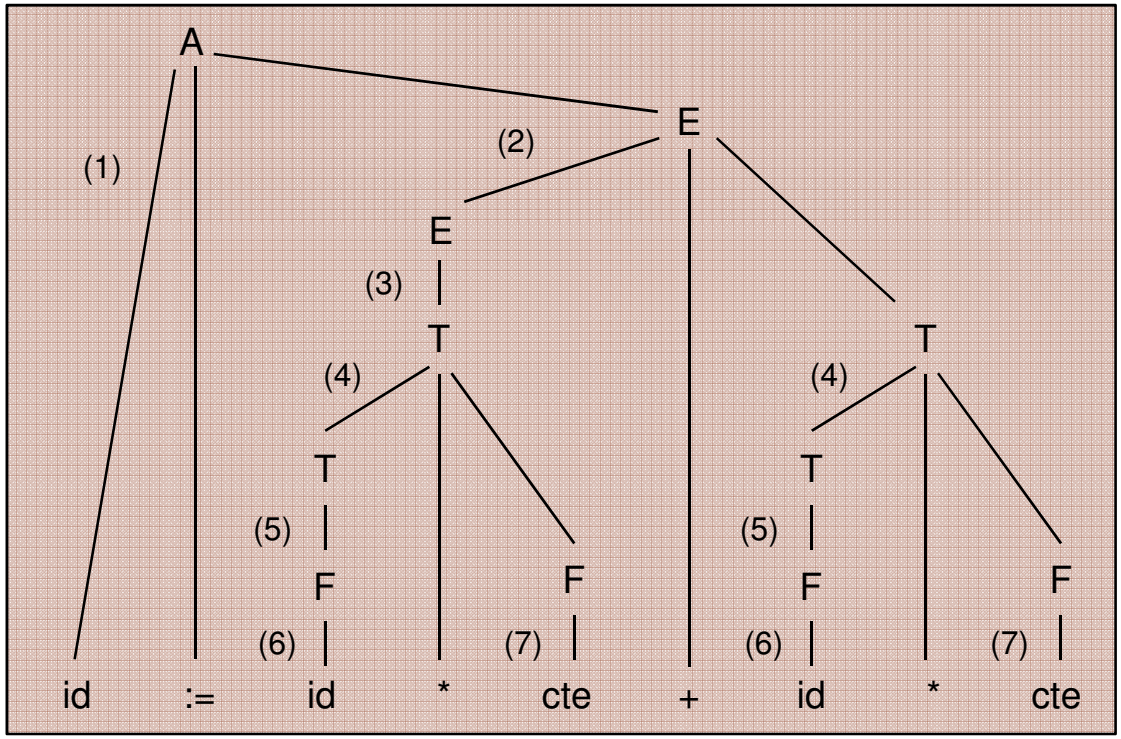
Parsing Ascendente
Ejemplo

precio := costo1 * 1.5 + costo2 * 1.2 → id := id * cte + id * cte

Gramática

1) A → id := E
2) E → E + T
3) E → T
4) T → T * F
5) T → F
6) F → id
7) F → cte

Árbol de Parsing



Lista de Reglas: 6 5 7 4 3 6 5 7 4 2 1

Gramáticas LR

- Parsing Ascendente Predictivo LR(0) y SLR(1) o LR(1) Simplificado
  - Intentan ***predecir*** qué reducciones aplicar sin necesidad de ver toda la entrada
  - Parsers shift/reduce
  - Conflictos
    - LR(0) → Falla
    - SLR(1) → Analiza un token más para decidir qué acción tomar
- LALR(1): Lookahead LR(1)
  - Usan reglas más precisas que SLR(1) para resolver conflictos.
  - **YACC usa esta técnica**

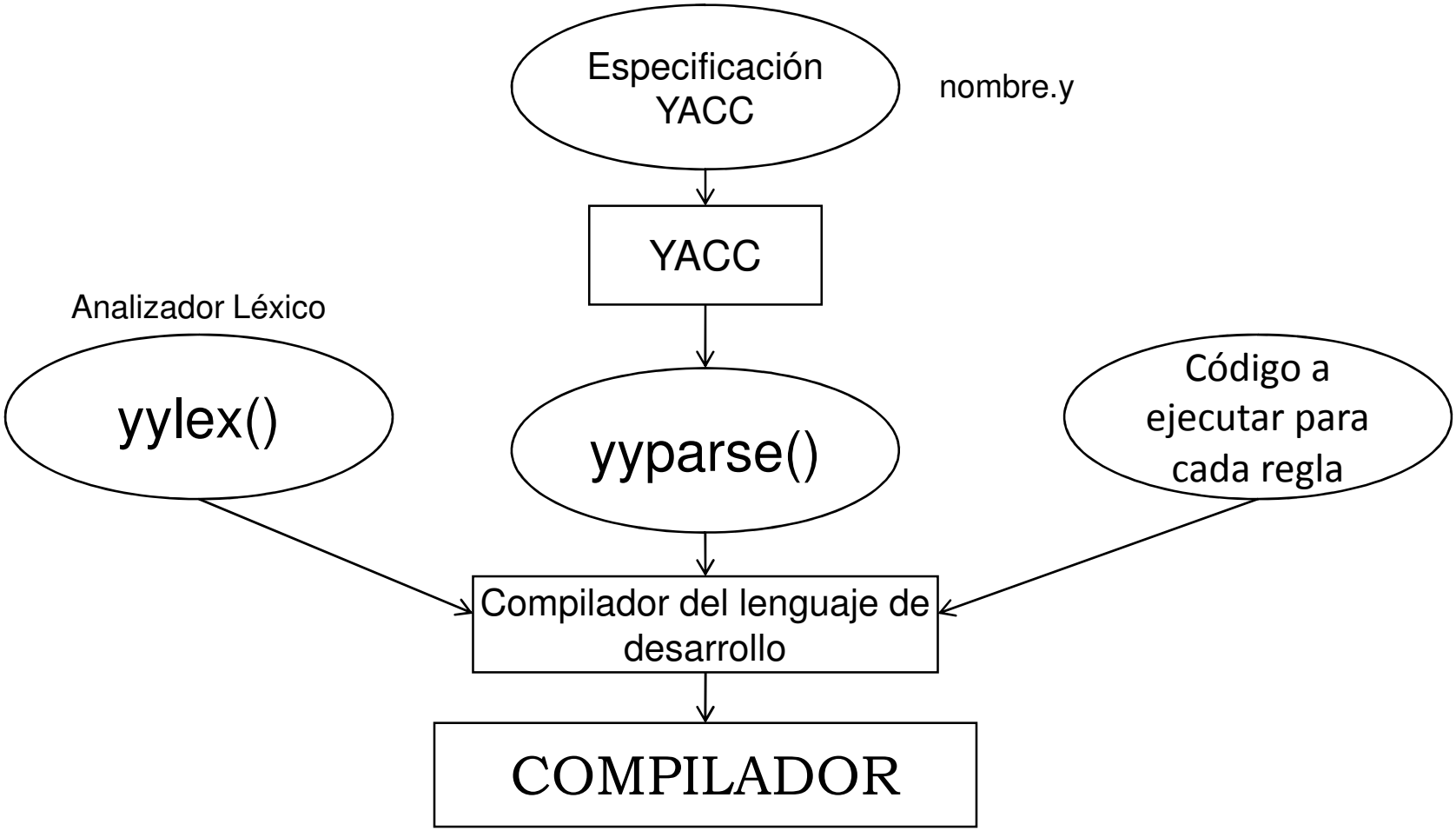
YACC

Yet Another Compiler Compiler

YACC

- YACC provee una herramienta general para analizar estructuralmente una entrada.
- Requiere una especificación que incluye:
  - Un conjunto de reglas que describen los elementos de la entrada (Gramática)
  - Un código a ser invocado cuando una regla es reconocida
  - Un Analizador Léxico que se encargue de proveer tokens

Yacc


Especificación YACC
nombre.y
YACC
Analizador Léxico
yylex()
yyparse()
Código a ejecutar para cada regla
Compilador del lenguaje de desarrollo
COMPILADOR

¿Cómo se usa YACC?

- Escribir una especificación YACC conteniendo la gramática. (.y por convención)
  - Ejemplo: gramatica.y

- Escribir un Analizador Léxico. El método o función léxica debe ser:
  - ***int yylex()***

  Nota: ***yyparse*** invoca a ***yylex*** cada vez que necesita un token.

- Ejecutar YACC con la especificación como parámetro, para generar el código fuente del parser.

  Byacc para C: **byacc gramatica.y**

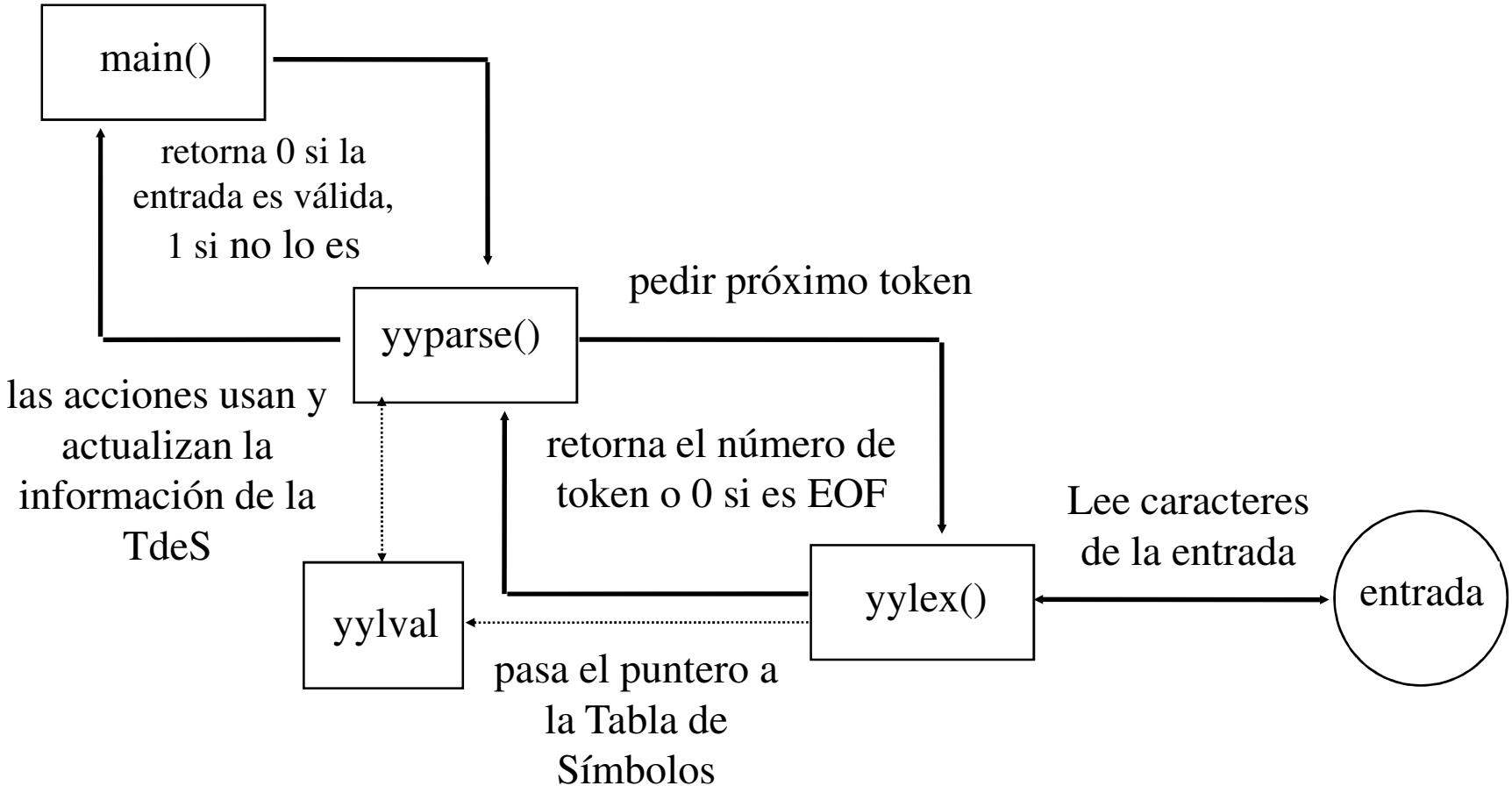
  La salida es un archivo llamado ***y_tab.c*** conteniendo la función ***yyparse***().

  Byacc para Java: yacc –J gramatica.y

  La salida es un archivo llamado ***Parser.java*** conteniendo el método ***yyparse***()

- Compilar y vincular los fuentes del Analizador Sintáctico, Analizador Léxico y todo otro código creado.

Interacción entre yyparse y el Analizador Léxico



Tokens

- El Analizador Léxico detecta un token y retorna un número de token al parser.
- El número de token es definido por un símbolo que el parser usa para identificar al token.
- Los números de token son definidos por YACC cuando procesa los tokens declarados en la especificación.
- Cada carácter ASCII es definido como un token cuyo número es su valor ASCII (0 a 256).
- Los tokens definidos por el usuario comienzan en 257.

Tokens

- Si, en la especificación para YACC, se incluye:

  %token ID CTE …

- En **yacc** para C, se generan sentencias #define para definir los números de tokens:

  #define ID 257

  #define CTE 258

  …

- Estas definiciones son ubicadas en el archivo y_tab.c, junto con la rutina **yyparse**, o pueden generarse en un archivo separado, llamado y_tab.h.
  - Para ello, se debe ejecutar byacc –d gramatica.y

Tokens

- Si, en la especificación para YACC, se incluye:

  %token ID CTE …

- En **byacc** para Java, en el archivo Parser.Java, junto con yyparse, se generan:

  public final static short ID=257;

  public final static short CTE=258;

  …

Tokens – **yylval**
(YACC para C)

- Para pasar información del token al parser, se utiliza una variable externa llamada **yylval.**
- En C, el tipo de **yylval** es int, por defecto.
- Se puede cambiar el tipo de **yylval** o,

  definir una unión de tipos de datos múltiples para **yylval:**

```
% union {
        int entero;
        char *cadena;
      }
```

Tokens – **yylval** (YACC para Java)

- YACC genera la clase pública ParserVal

```
public class ParserVal
{
  public int ival;
  public double dval;
  public String sval;
  public Object obj;
  public ParserVal(int val)
  {
    ival=val;
  }
  public ParserVal(double val)
  {
    dval=val;
  }
  public ParserVal(String val)
  {
    sval=val;
  }
  public ParserVal(Object val)
  {
    obj=val;
  }
}//end class
```

Especificación YACC

Formato:

declaraciones

%%

reglas gramaticales

%%

código

Autómata de Pila

- Los autómatas finitos son suficientes para el Analizador Léxico.
- Los A.F. no son suficientes para un Analizador Sintáctico, porque son incapaces de recordar el *estado anterior*.
- YACC genera un autómata de pila.

Autómata de Pila

- Tiene un número finito de estados, una función de transición, una entrada, y está equipado con una ***pila***.
- La función de transición trabaja sobre el estado actual, el elemento en el tope de la pila, y el token de entrada actual, produciendo un nuevo estado.
- Permite a YACC reconocer gramáticas LALR (LookAhead Left Recursive) o LR(1).

Autómata de Pila

- El estado actual es siempre el del tope de la pila.
- Inicialmente, la pila contiene sólo el estado 0 y no se ha leído ningún token.
- El autómata tiene 4 acciones disponibles:
  - **shift**,
  - **reduce**,
  - **accept**,
  - y **error**.

Ejemplo: Especificación YACC

```
%token    A   B   C
%%
lista :       inicio   fin
              ;
inicio        :        A   B
              ;
fin           :        C
```

Ejemplo: Autómata

```
state 0
        $accept  :  _lista  $end

        A  shift  3
        .  error

        lista  goto  1
        inicio  goto  2

state 1
        $accept  :  lista_$end

        $end  accept
        .  error

state 2
        lista  :  inicio_fin

        C  shift  5
        .  error

        fin  goto  4

state 3
        inicio  :  A_B

        B  shift  6
        .  error

state 4
        lista  :  inicio  fin_      (1)

        .  reduce  1

state 5
        fin  :  C_                  (3)

        .  reduce 3

state 6
        inicio  :  A  B_            (2)

        .  reduce  2
```

Autómata de Pila

- Para obtener el autómata, se debe ejecutar yacc con la opción –v

  byacc –v gramatica.y

Especificación YACC

Formato:

declaraciones

%%

reglas gramaticales

%%

código