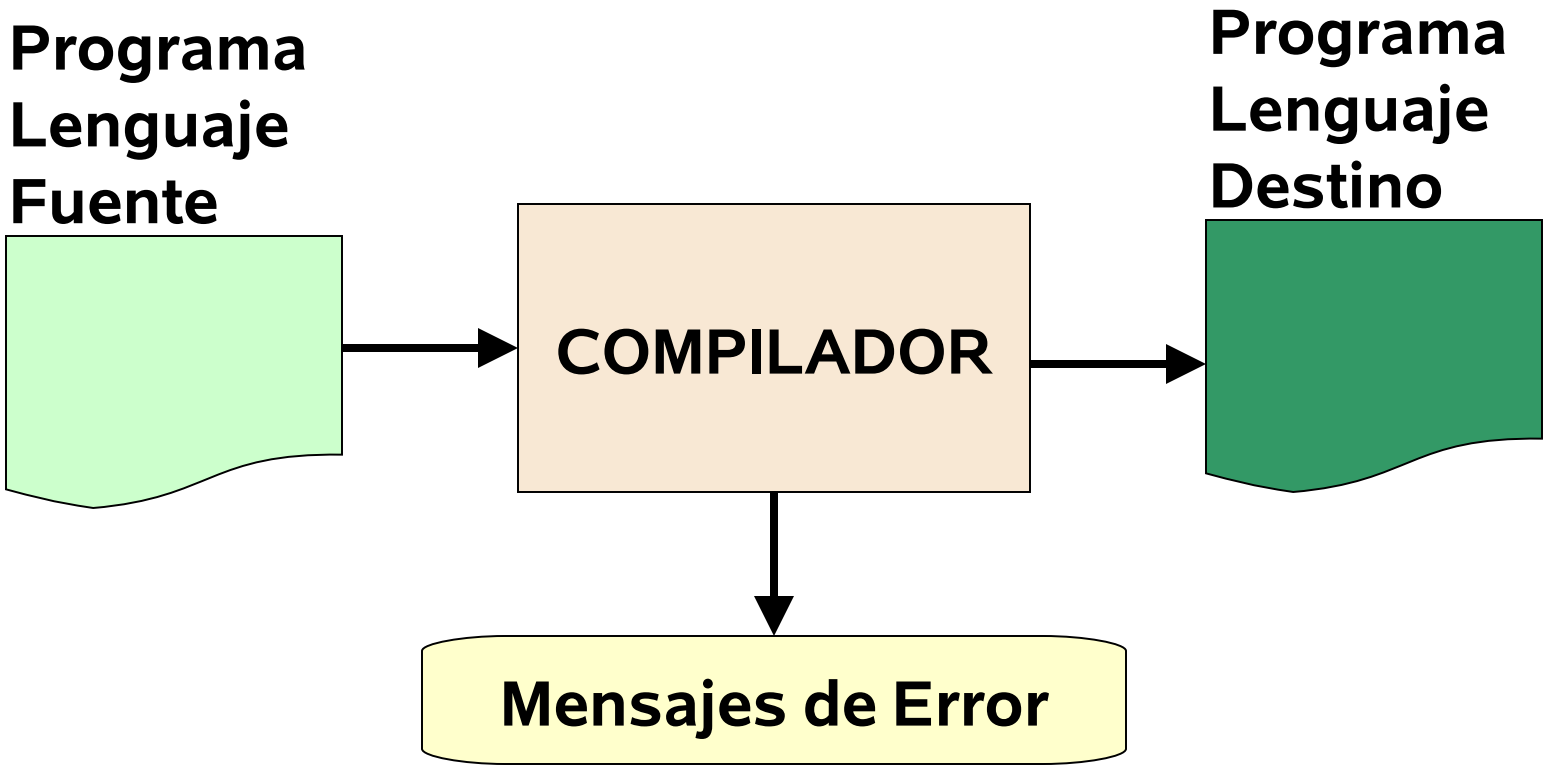
300CIG007

Computabilidad y Lenguajes Formales: Compiladores e Interpretes

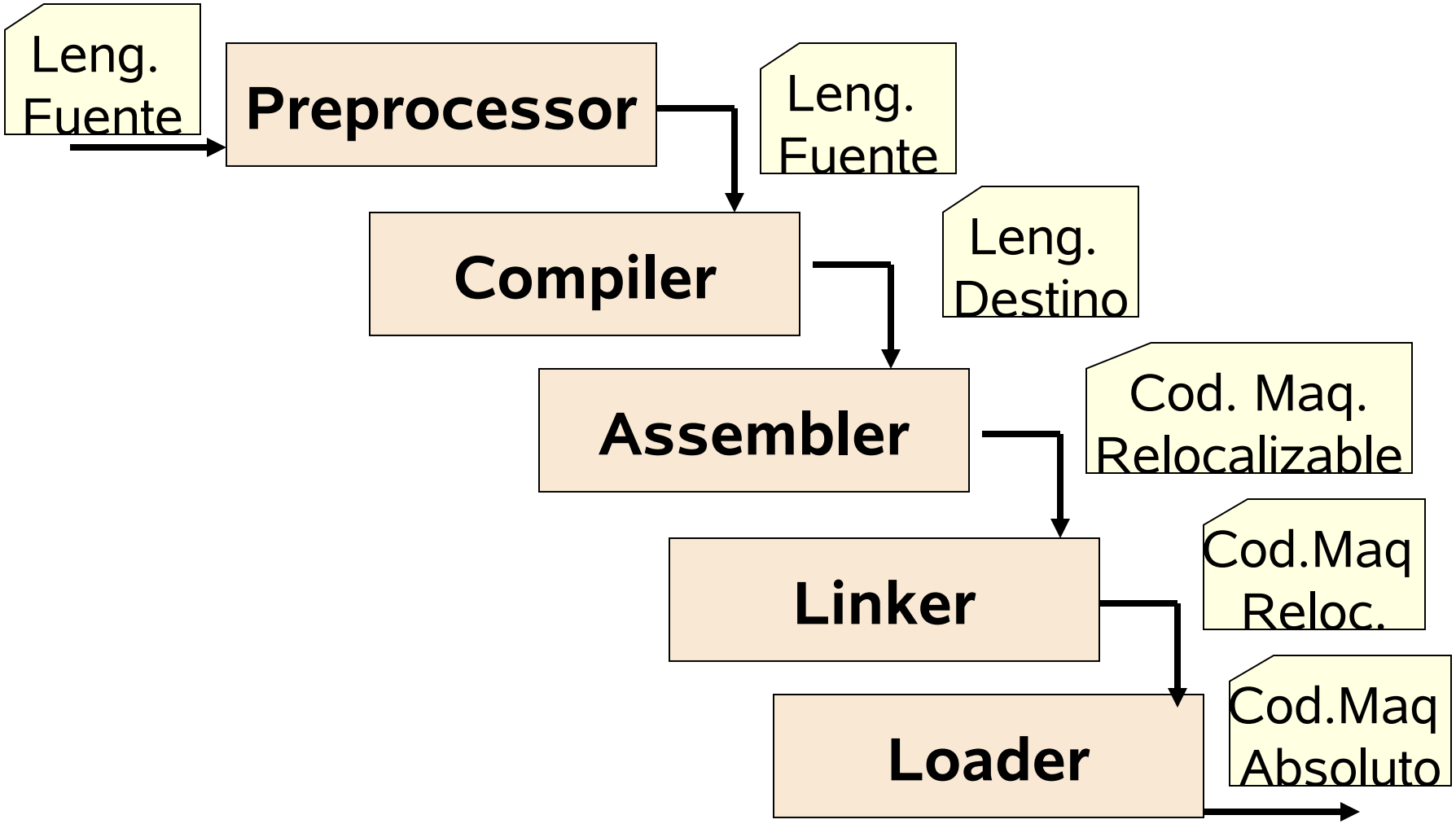
Pontificia Universidad Javeriana Cali
Ingeniería de Sistemas y Computación
Prof. Gloria Inés Alvarez V.

Qué es un COMPILADOR?

Qué es un COMPILADOR?



Contexto de un Compilador

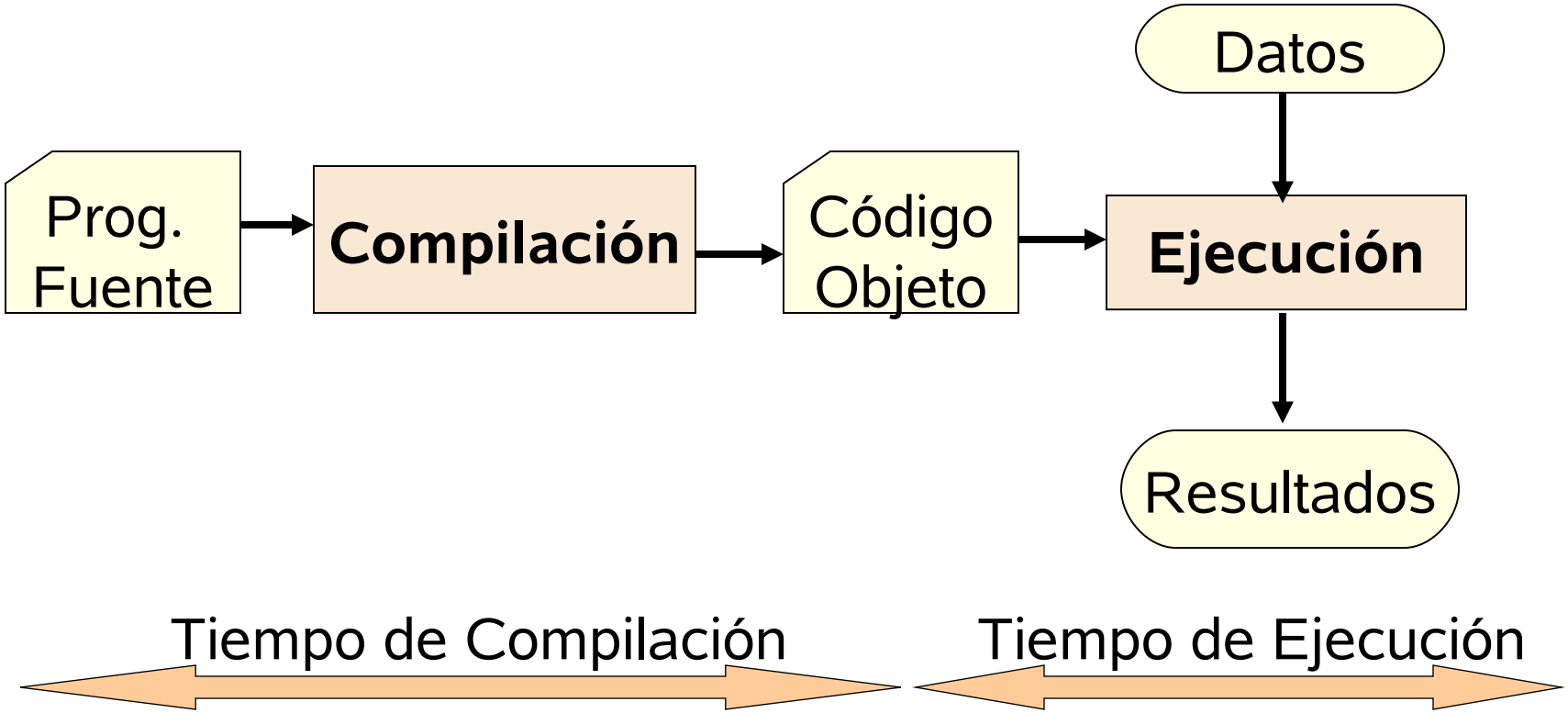


Implementación de Lenguajes de Programación: Compilador - Interprete

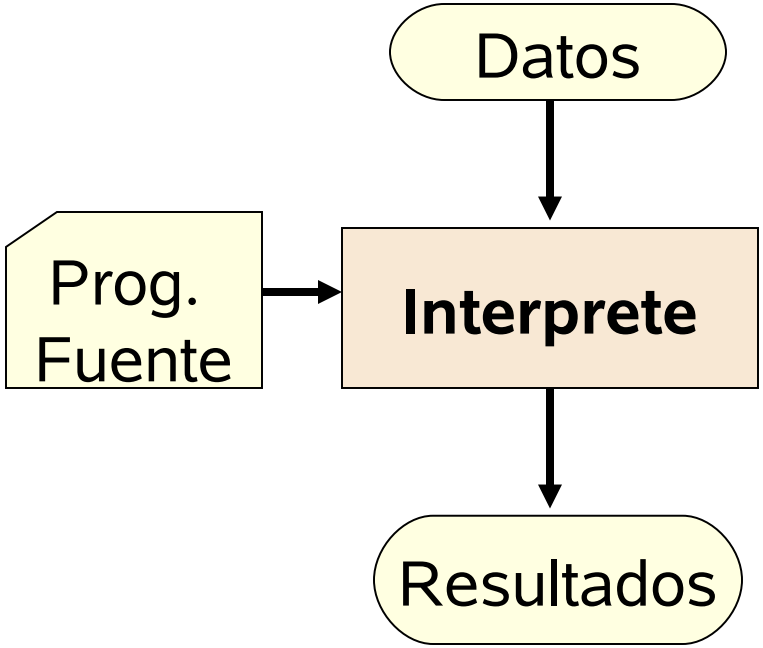
- Interprete: no realiza la traducción, desarrolla las operaciones que se especifican en el programa fuente. Procesa el programa y la entrada simultáneamente.

- Compilador.
  - Tiempo de Compilación
  - Tiempo de Ejecución

Compilador


Datos
Prog. Fuente
Compilación
Código Objeto
Ejecución
Resultados
Tiempo de Compilación
Tiempo de Ejecución

Interprete



Tarea del Compilador

```
int a,b;
main ()
{
  a = 5;
  b = a*a+1;
}
```

→ **COMPILADOR** →

```
STORE  #5,0
LOAD   0,R0
MUL    0,R0
ADD    #1,R0
STORE R0,4
```

Como lo hace?

How does a compiler work? (1/5)

(Tomado de las notas de profesor J. Schaeffer - U.Alberta)

- A compiler performs its task in a manner "similar" to how a human approaches the same problem.
- Consider the following sequence of line segments:

  Wr i t **e** a C o **m p** i l e r

- We all understand what it means, but how did you arrive at that conclusion?
- Can you describe the processes by which you attach some meaning to this "random" series of lines?

How does a compiler work? (2/5)

(Tomado de las notas de profesor J. Schaeffer - U.Alberta)

- Unconsciously, we carry out a number of processes, including:
  - Recognize patterns (characters) that have some meaning to us. This requires a set of recognizable symbols (alphabet, mathematical signs, punctuation, etc.). In the example, we have 14 explicit patterns and 2 implicit (blanks).
- Group characters into logical entities (words). In the example, treat it as 3 words.
- This activities correspond to the **lexical analysis** phase of a compiler.

How does a compiler work? (3/5)

(Tomado de las notas de profesor J. Schaeffer - U.Alberta)

- Having identified the words, associate a meaning with each one. Look them up in a dictionary for their meaning and some attributes (e.g. part of speech).
- Compiler has a **symbol table** (dictionary) to save and retrieve words and their attributes.
- Check that the words form a structurally correct sentence. The example is correct, but what about:

  "compiler a write".
- This sentence obviously violates some "rules" that we know for constructing correct sentences.
- Checking the structure of sentences is the **syntax analysis** phase.

How does a compiler work? (4/4)

(Tomado de las notas de profesor J. Schaeffer - U.Alberta)

- Check that the combination of words makes sense. For example, the sentence:

  "dig a compiler"

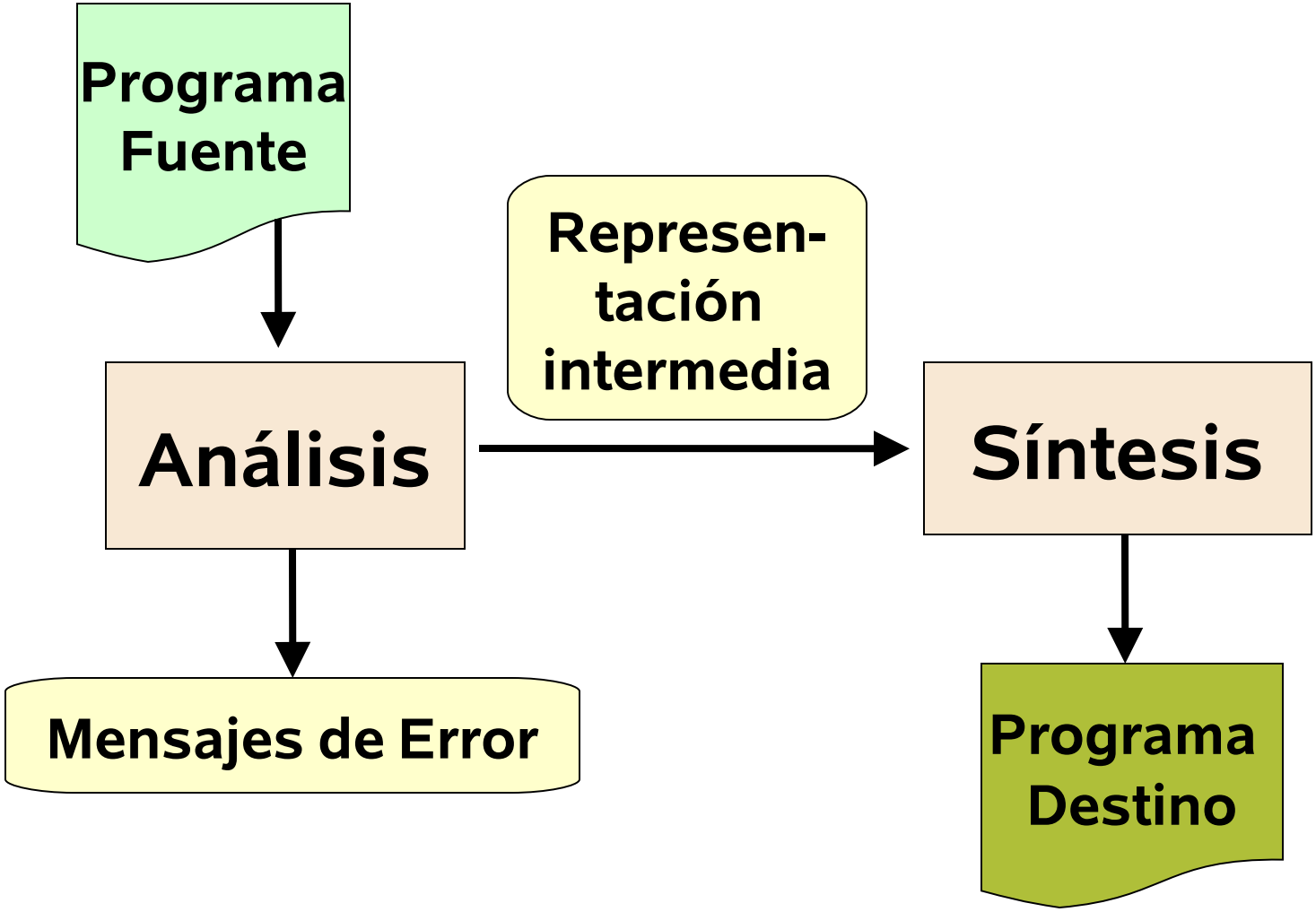
  may be syntactically correct, but its meaning is questionable.
- Verifying that a sentence is meaningful is done in the **semantic analysis** phase.

How does a compiler work? (5/5)

(Tomado de las notas de profesor J. Schaeffer - U.Alberta)

- Now we have a correct, meaningful sentence. Plan what you have to do to accomplish the task. In a compiler, this means to generate the instructions to carry out the task.
- The code generation phase of a compiler determines the sequence of instructions to be carried out.
- Finally, in a compiler, do an optional retrospective look at the code generated to see if it can be improved.
- The code improvement or code optimization phase of a compiler tries to reduce the effort required to carry out the program.
- Execute. Do it!

Modelo de Compilación

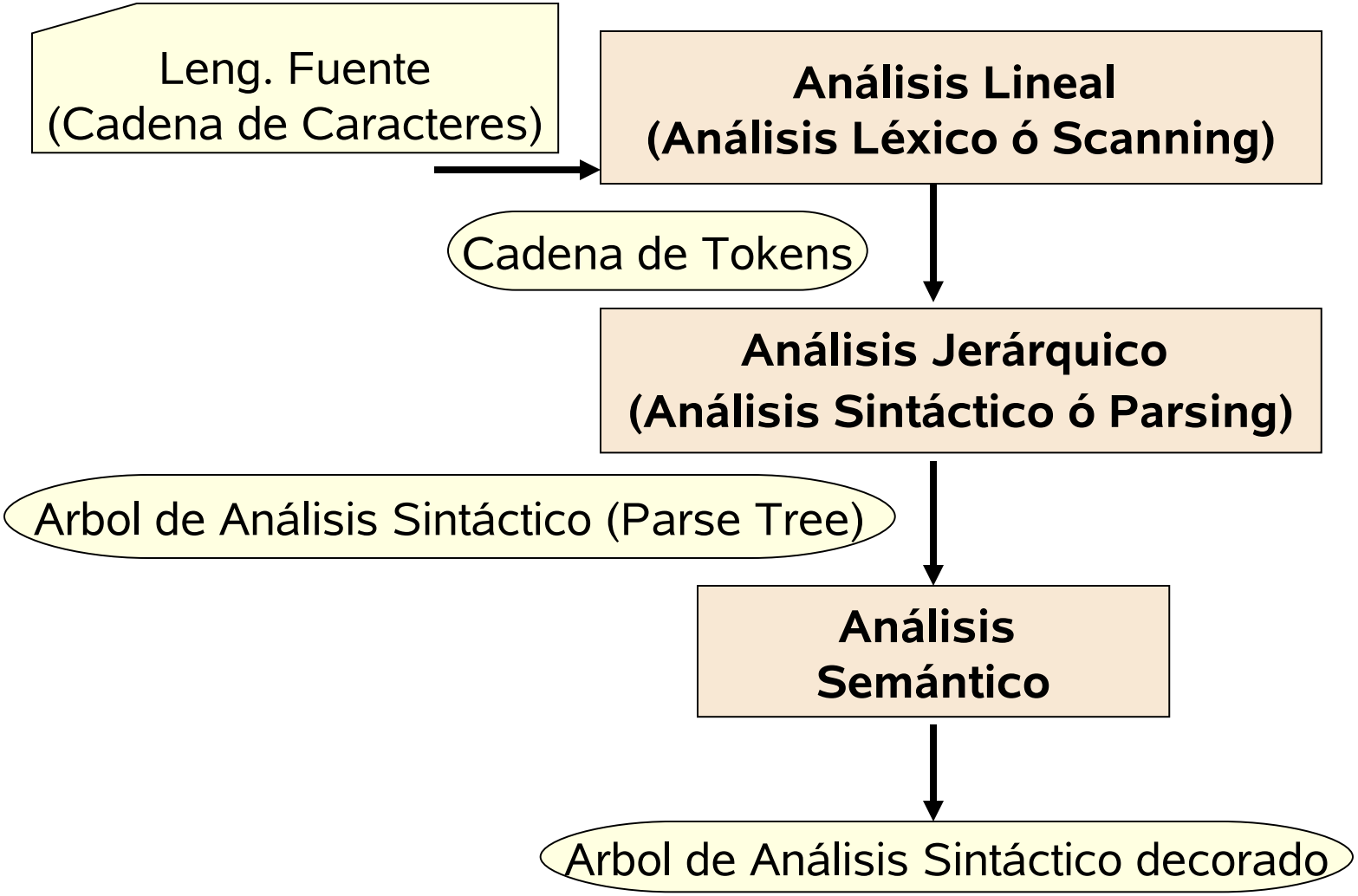


Modelo de Compilación

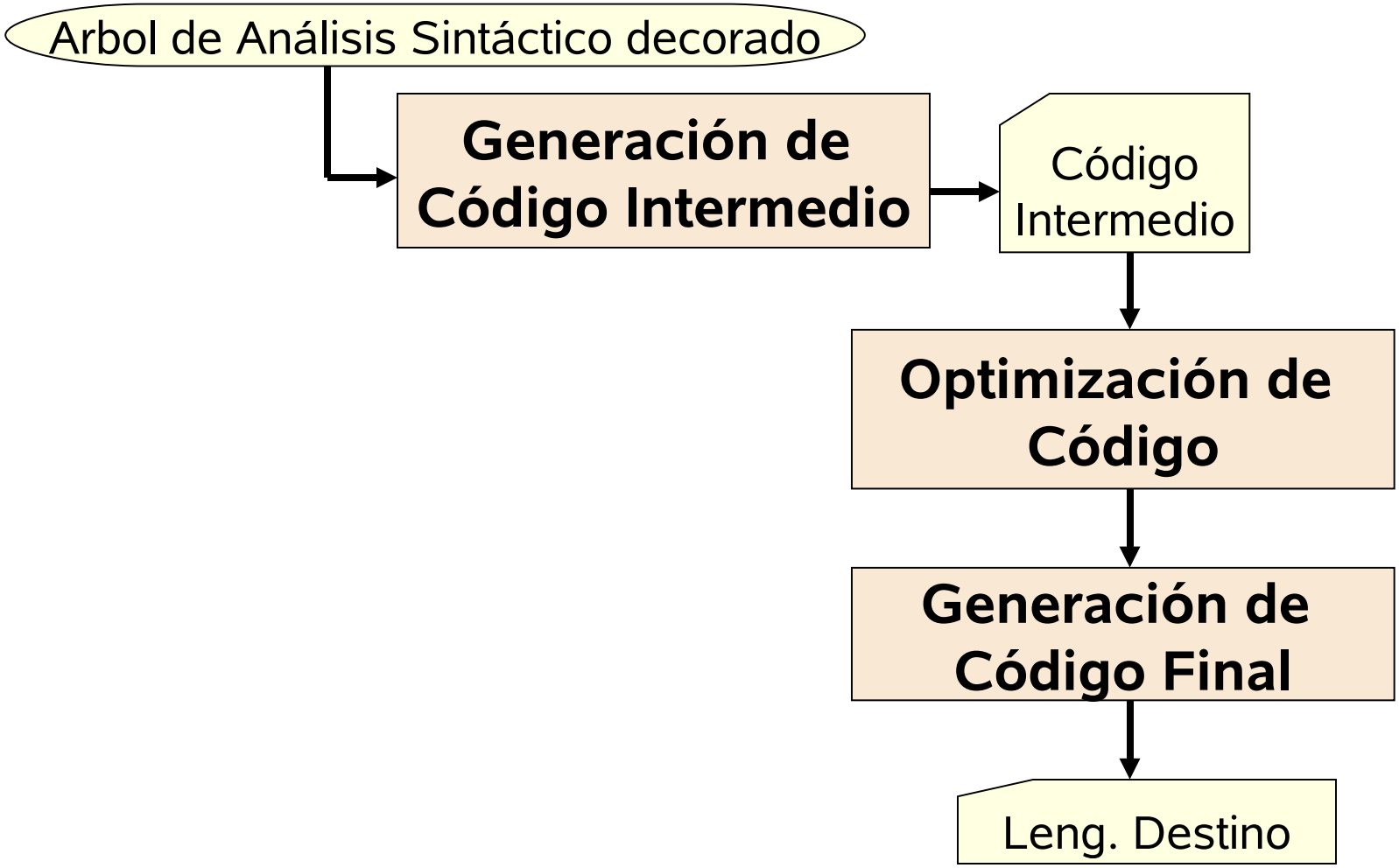
- Análisis: divide el programa fuente en las partes que lo forman y crea una representación intermedia

- Síntesis: construye el programa destino a partir de la representación intermedia

Fases de Análisis



Fases de Síntesis



Además

- Durante todas las fases se usa una Tabla de Símbolos
- Durante las fases de análisis se hace Detección, Reporte y Recuperación de Errores.

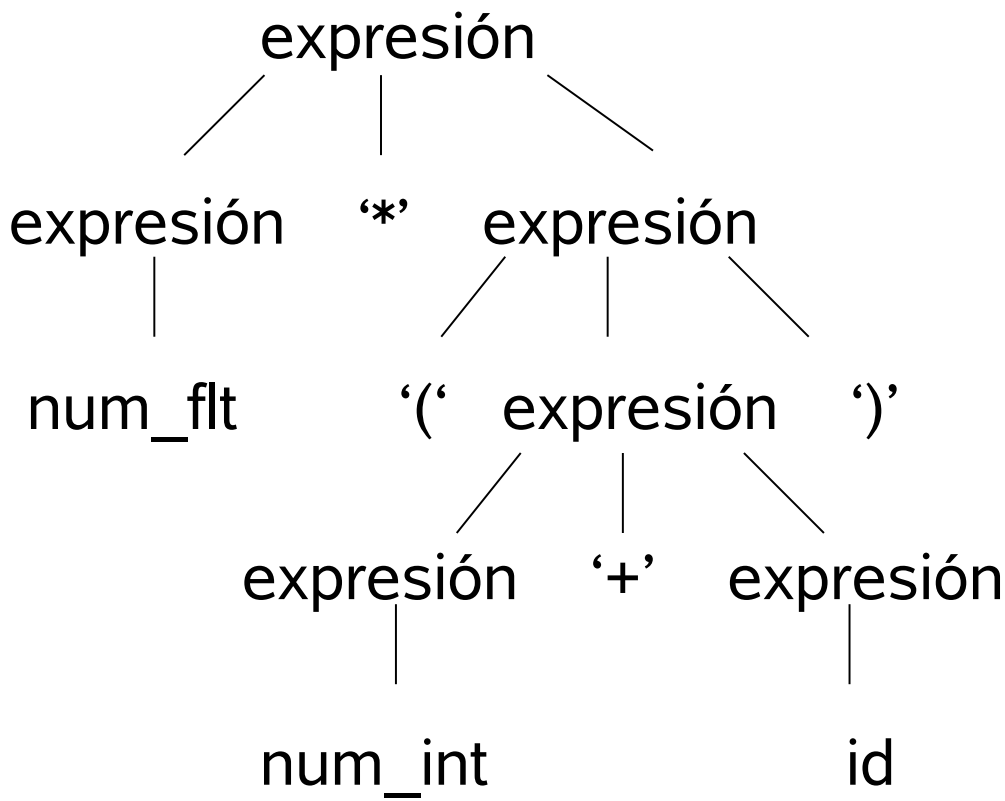
Análisis Léxico (Scanner)

.....	1	2	.	5		*		(	5	+	v	a	l	o	r	)	;	\n	.....

Número flotante (12.5)
Espacio
Operador de multiplicación
Espacio
Paréntesis abierto
Número entero (5)
Operador de suma
Identificador (valor)
Punto y coma
Salto de línea

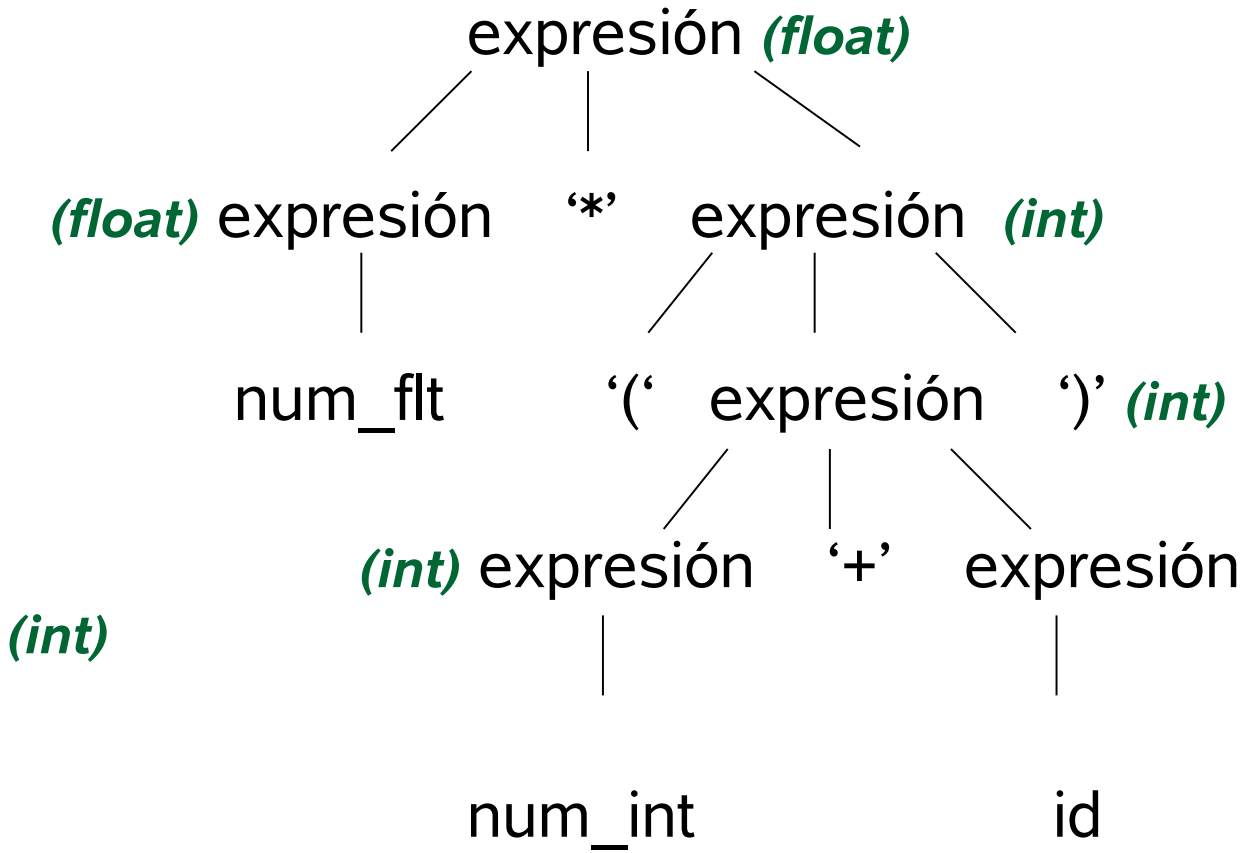
Análisis Sintáctico (Parser)

Num_flt ‘*’ ‘(‘ num_int ‘+’ id ‘)’ ‘;’



Análisis Semántico

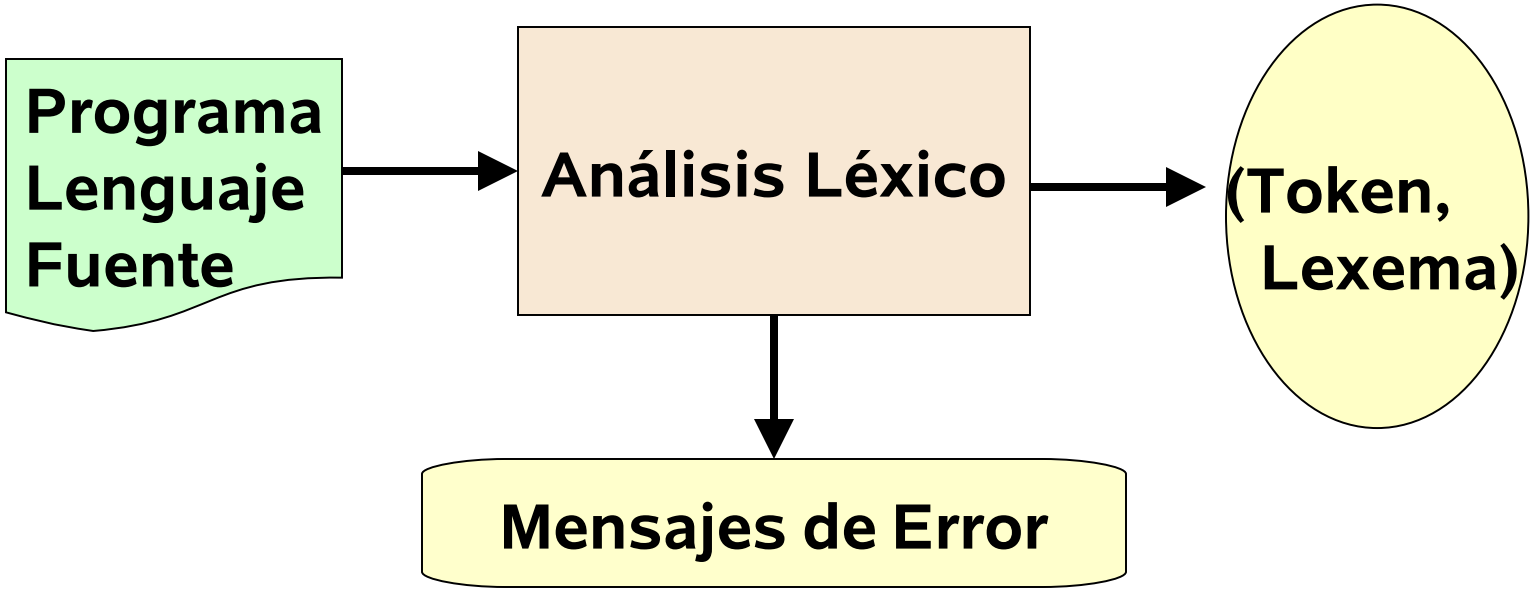
Num_flt ‘*’ ‘(‘ num_int ‘+’ id ‘)’



Herramientas y técnicas

Fase	Definición Formal	Máquina que lo reconoce	Herramien-ta
Léxico	Expresiones Regulares	Autómata Finito	Generador de Scanners
Sintáctico	Gramáticas Libres de Contexto	Autómata de Pila	Generador de Parsers
Semántico		Traducciones dirigidas por Sintaxis	Generador de Parsers
Generación Código		Traducciones dirigidas por S.	Generador de Parsers

Análisis Léxico



Lee los caracteres del programa fuente de izquierda a derecha, y los agrupa en tokens

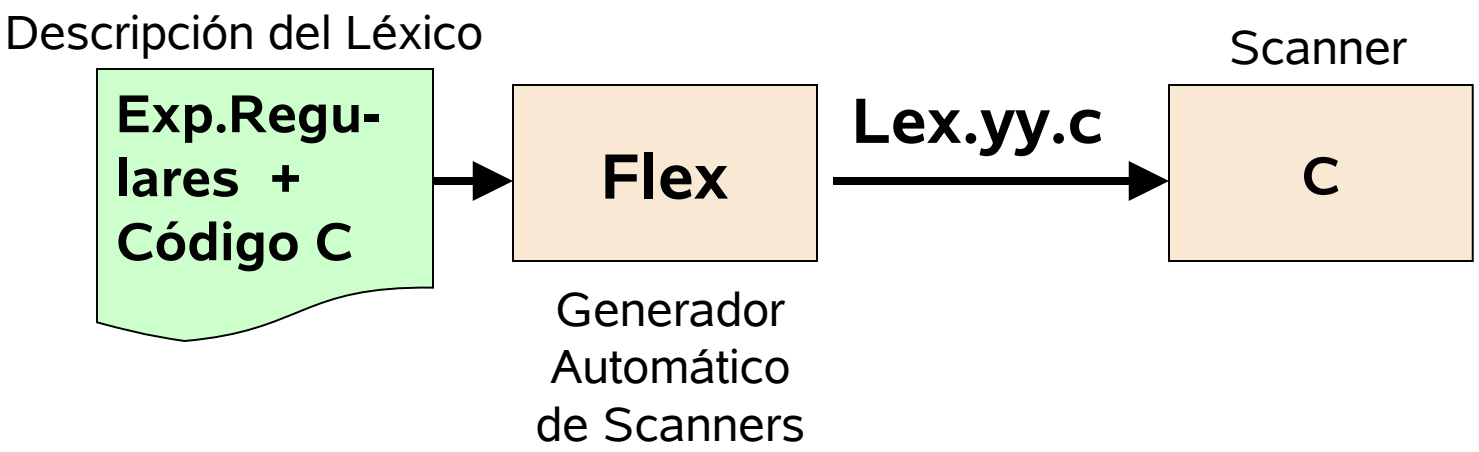
Tokens y Lexemas: Ejemplos

Token	Token #	Lexema
If	10	‘if’
parentesisIzq	12	‘(‘
Identificador	1	‘valor’
opSuma	30	‘+’
puntoComa	48	‘;’
Menor	56	‘<‘
menorIgual	57	‘<=‘

Funciones del Analizador Léxico

- Convierte el programa fuente en una cadena de tokens
  - Para reconocer el token usa un patrón, una regla que describe como se forman las cadenas que corresponden a un token.
- Salta comentarios y espacios en blanco (tabuladores, saltos de línea...)
- Tener el registro de la línea del archivo fuente que está siendo analizada
- Genera mensajes de error léxico, y se recupera del error
- Convierte los valores literales al tipo que corresponda
- Si la entrada debe obedecer a un formato, verifica el formato Ej. Fortran, Cobol

Generadores Automáticos de Scanners: Flex



Lex.yy.c contiene la rutina yylex(), que cada vez que es invocada analiza la entrada para encontrar el siguiente token.

Ejemplo del archivo de entrada a Flex

```
%{
    #include "token.h"
    #include "lexico.h"
%}
LETRA               [a-zA-Z]
%%
if                  { return (TIF); }
then                { return (TTHEN); }
else                { return (TELSE); }
true|false          { return (TVALOG); }
{LETRA}+            { return (TID); }
"{"                 { return (TLLA); }
"}"                 { return (TLLC); }
"="                 { return (TASIG); }
";"                 { return (TPC); }
[ \t]+              /* obvia espacios en blanco */
[\n]                { nro_lineas++; }
.                   { printf (" Linea %d : Caracter desconocido  %s\n",
                              nro_lineas, yytext); }
%%
```

Ejemplo del archivo de entrada a Flex

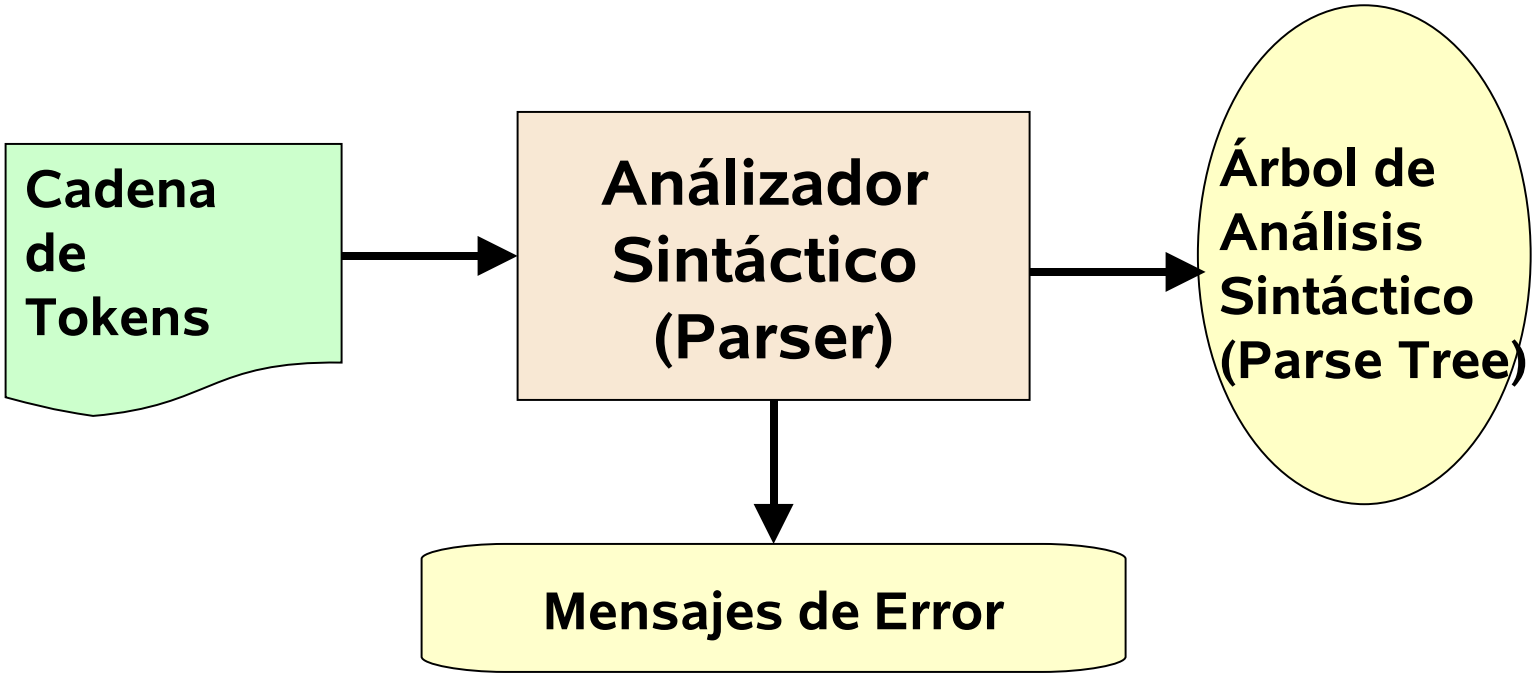
- Token.h

```
#define   TID       1
#define   TIF       2
#define   TTHEN     3
#define   TELSE     4
#define   TLLA      5
#define   TLLC      6
#define   TASIG     7
#define   TPC       8
#define   TVALOG    9
```

- Lexico.h

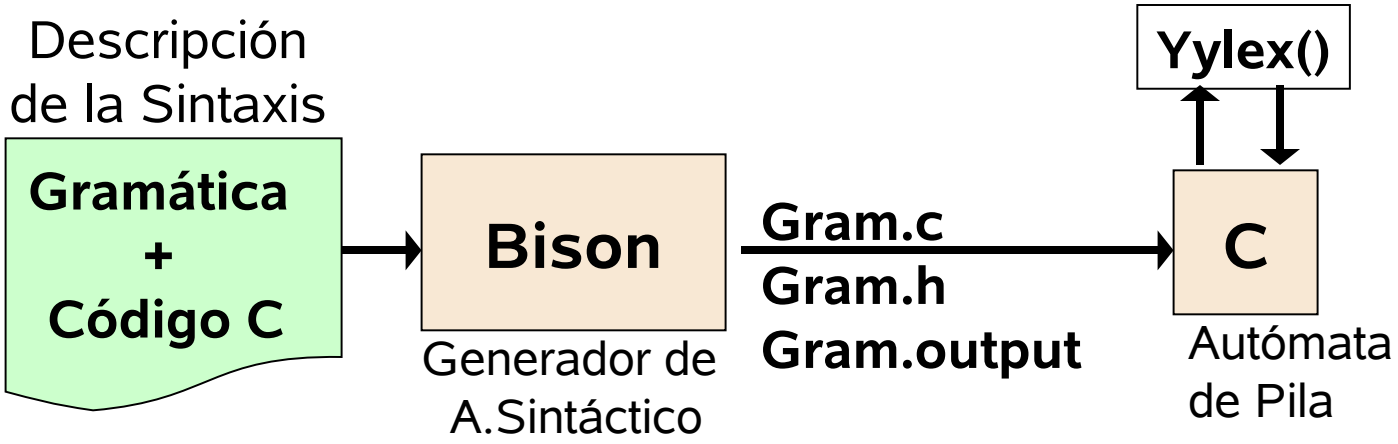
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
// Variables
int nro_lineas;

extern FILE  *yyin;
extern char *yytext;
// Prototipo de las funciones
extern int yylex();
```

Análisis Sintáctico



Obtiene la cadena de tokens del analizador léxico y verifica que puede ser generada por la gramática que describe el lenguaje fuente

Generadores Automaticos de Analizadores Sintacticos: Bison

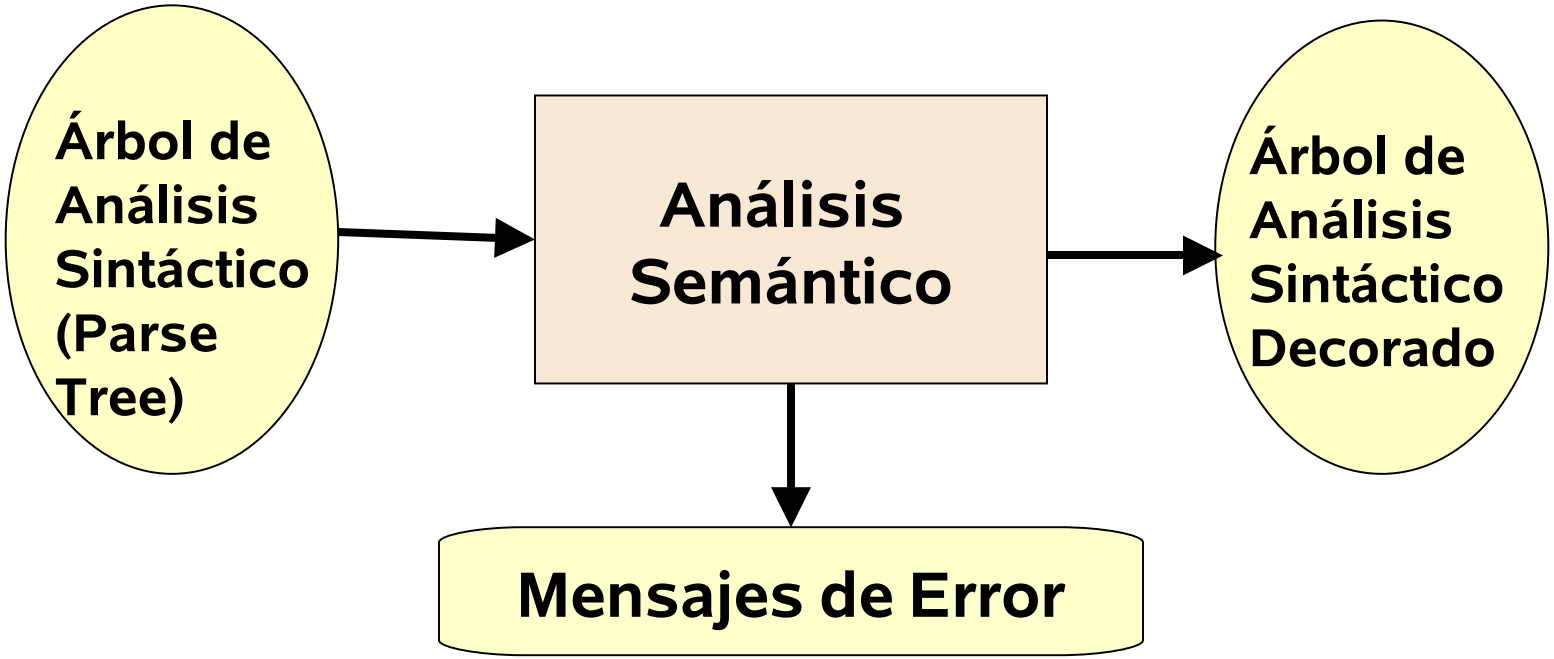


- El archivo ".c" contiene el código del analizador sintáctico, en la rutina yyparse()
- El archivo ".h" contiene la definición de constantes para los tokens usados en la gramática
- El archivo ".output contiene información sobre la construcción del autómata, número asignado a cada regla de producción, estados generados, conflictos.

Ejemplo de un archivo de entrada a Bison

```
%{
      #include “lex.yy.c"
      int yyerror(char *s);
%}
%token      TINTEGER
%token      TREAL
..
%left       TMAS, TMENOS
%left       TPOR, TDIV
%%
programa : TPROGRAMA TID TPC declaraciones principal
           | error declaraciones  { yyerror(“Error en nombre de programa”); }
;
declaraciones: decConstantes
           | decVariables
           | decFunciones
;
%%
Int yyerror(char *texto)
{    printf(“Linea %d: Error sintactico, %s\n”, nlineas, texto);
     return 1;
}
```

Análisis Semántico



Análisis Semántico: Comprobación Estática

- Comprobación de tipos: reporta errores cuando un operador se aplica a operandos incompatibles

- Comprobación de unicidad: situaciones en las que un nombre no puede ser declarado más de una vez.

- Comprobación de nombres relacionados: si el mismo nombre debe aparecer mas de una vez en la construcción.

  Ejemplo: el nombre de la función debe aparecer al comienzo y final de la declaración de la misma

- Comprobación de control de flujo: las instrucciones que permiten salir de una construcción deben tener un lugar a donde transferir el control. Ej. break

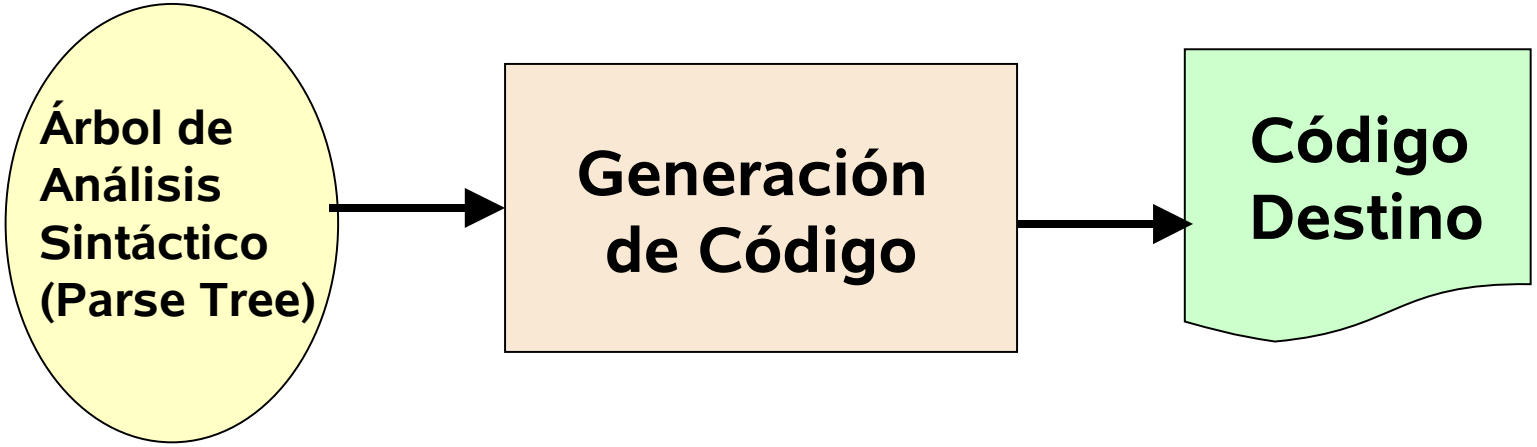
Definiciones Dirigidas por Sintaxis

- Es una generalización de las gramáticas libres de contexto CFG, en la cual se asocia un conjunto de atributos a cada símbolo de la gramática.
- Los atributos pueden ser Sintetizados o Heredados.
- Se puede pensar en cada nodo del árbol de análisis sintáctico como un registro, en el cual cada atributo corresponde con un campo.
- El valor del atributo en un nodo del árbol, se determina por la regla semántica asociada por la producción usada en ese nodo.
- El valor de un atributo sintetizado se calcula a partir de los valores de los atributos de los hijos del nodo.
- El valor de un atributo heredado se calcula a partir de los valores de los atributos de los hermanos y el padre del nodo.
- Un árbol que muestra los valores de los atributos es un árbol anotado o decorado.

Ejemplo: Definición Dirigida por Sintaxis

Reglas de Producción	Reglas Semánticas
$E' \rightarrow E$	{ print (E.valor); }
$E \rightarrow E_1 + T$	{ E.valor = E_1.valor * T.valor; }
$E \rightarrow T$	{ E.valor = T.valor; }
$T \rightarrow T_1$ * num	{ T.valor = T_1.valor * num.valor; }
$T \rightarrow$ num	{ T.valor = num.valor; }

Ejemplo de un archivo de entrada a Bison

```
%{   /*  .... Declaraciones en C  */
%}
%token  ...
%union
{   char    nombre[100];
    struct  tipo  tipo;
    int       codtipo;
};
%type    <tipo>  literal;
%type   <codtipo>  expresion;
%%
programa : ...  ;
literal: TLITINT   { $$.codtipo = 0; $$.valorint = $1.valor; }
        |  TCHAR   { $$.codtipo = 1; $$.valorch = $1.valor; }
expresion: expresion TMAS expresion
         {    if  $1 <> ERR && $3 <> ERR
                  if $1 == $3 && $1 == 0
                      $$ = $1;
                  else
                      printf("Error Semantico en la línea %d: Tipos o validos
      para la suma", nrolinea);
         }
......
;
%%
.....
```

Generación de Código