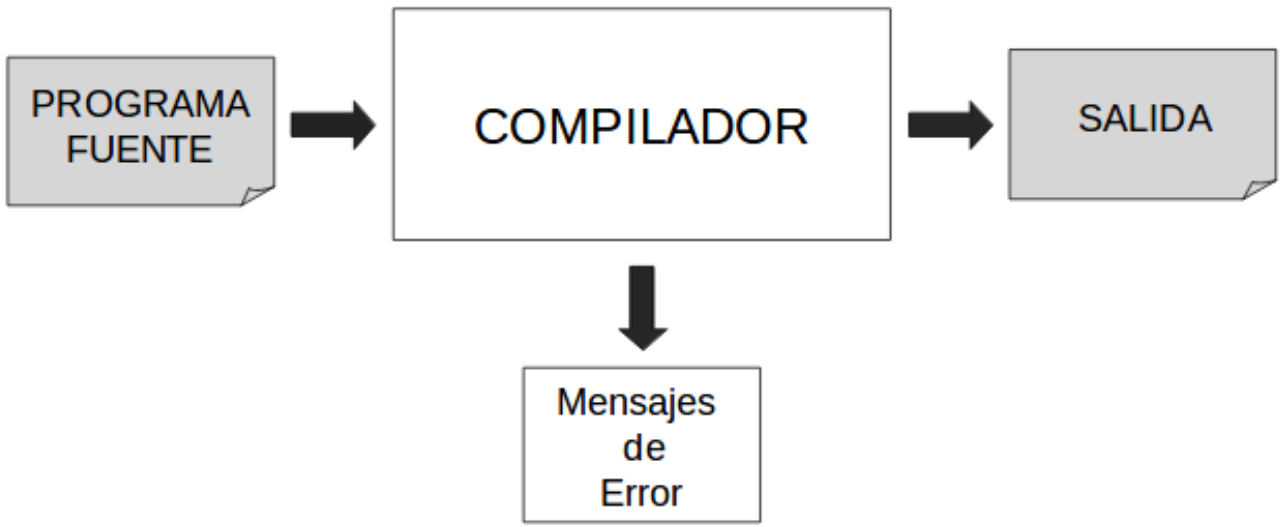
Arquitecturas Computacionales

Estructura general de un compilador

Facultad de Ingeniería / Escuela de Informática
Universidad Andrés Bello, Viña del Mar.

Estructura general de un compilador

- Un compilador es un programa que traduce un programa escrito en lenguaje fuente y produce otro equivalente escrito en lenguaje destino



Entrada de un compilador

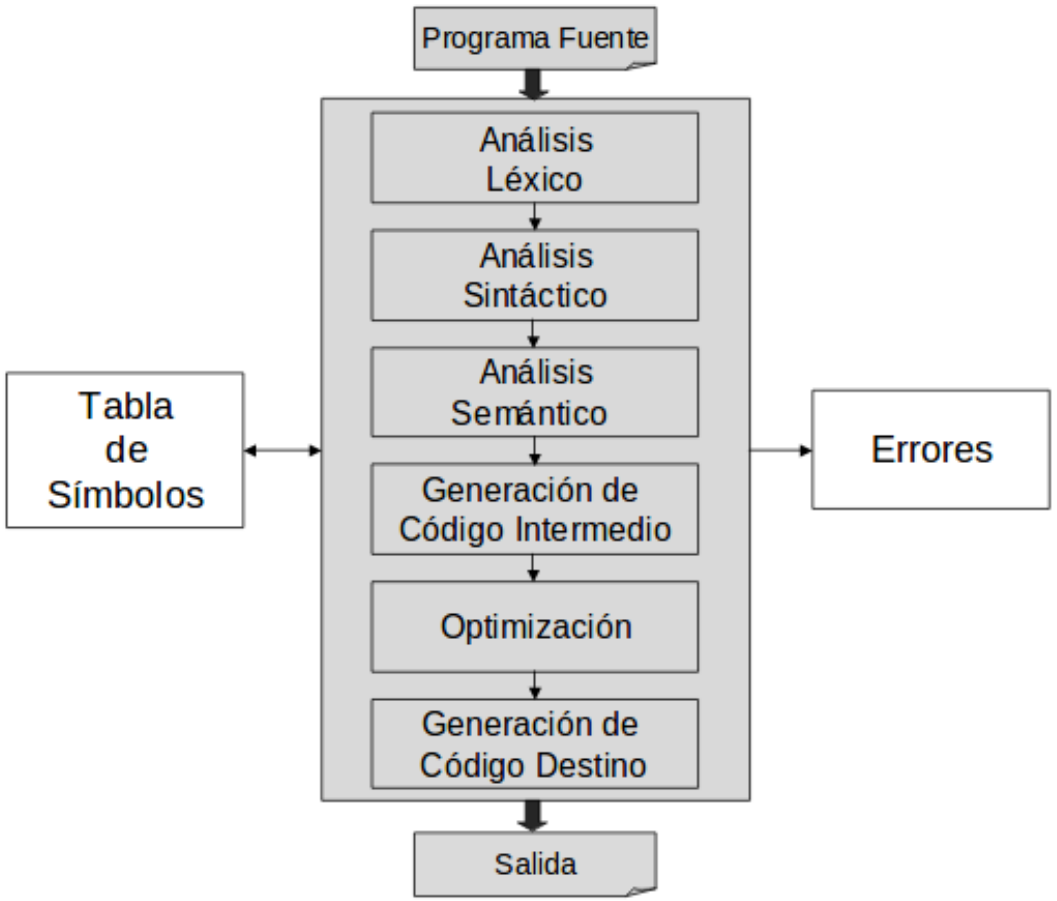
- Lenguaje de alto nivel (p.ej. C, C++)
- Lenguaje especializado para cierto fin

UNIVERSIDAD ANDRES BELLO

Salida de un compilador

- código *assembler* (debe ser ensamblado y vinculado)
- código binario (debe ser vinculado con librerías correspondientes)
- otro lenguaje de alto nivel

Fases de la compilación



Análisis (Front End)

Fases que dependen del lenguaje de origen

- análisis léxico
- análisis sintáctico
- análisis semántico (estático)
- creación de tabla de símbolos
- generación de código intermedio
- algoritmo de optimización
- manejo de errores del análisis

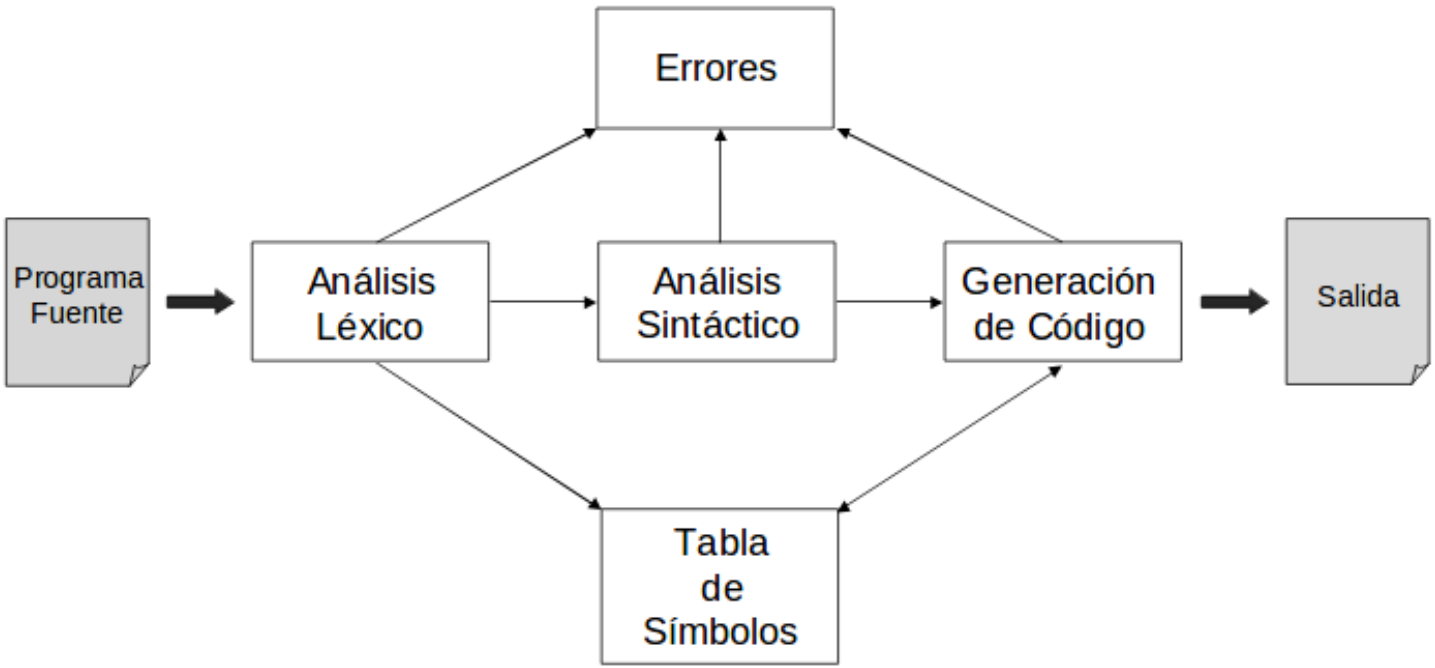
Síntesis (Back End)

Fases que dependen de la máquina de destino

- generación de la salida
- optimización
- operaciones sobre la tabla de símbolos
- manejo de errores de síntesis

UNIVERSIDAD ANDRES BELLO

Fases de la compilación



Fases de la compilación

frecuentemente hay pre-procesadores para

- eliminar comentarios
- incluir archivos
- expandir macros
- compilar condicionalmente
- reemplazar constantes simbólicas

UNIVERSIDAD ANDRES BELLO

Análisis léxico

- lee el programa de origen
- elimina espacios en blanco, tabulaciones y saltos de línea
- elimina comentarios
- agrupa los caracteres en unidades denominadas **tokens**

Análisis léxico

la interacción entre análisis léxico y sintáctico puede ocurrir de distintas maneras:

- ambas se ejecutan en modo batch
- ambas son concurrentes
- ambas son rutinas del generador de código
- el análisis léxico es una rutina del análisis sintáctico

Análisis léxico (ejemplo)

```
if Plazo >= 30
        Tasa  = Base + Recargo / 100
   else Tasa  = Base
```

Análisis léxico (ejemplo)

[if] [Plazo] [>=] [30]
[then] [Tasa] [=] [Base] [+] [Recargo] [/] [100]
[else] [Tasa] [=] [Base]

Análisis léxico

Los datos presentes pueden ser clasificados como

- Palabras reservadas: `if`, `then`, `else`
- Operadores: `+`, `>=`, `=`
- Cadenas de múltiples caracteres: *identificador, constante*

Análisis léxico

Así, los tokens nos permiten diferenciar la cadena de caracteres que representan

- La cadena de caracteres es el valor léxico (o lexema)
  - existen tokens que se corresponden con un único lexema (p.ej. `if`)
  - existen tokens que pueden representar lexemas diferentes (p.ej. `plazo`)

Análisis léxico

- el análisis léxico hace una correspondencia entre cada token y un número entero
- entrega al análisis sintáctico los tokens
- cuando un token corresponde a más de un lexema, se entrega el par token-atributo

Análisis léxico (ejemplo)

Token	Identificación del token
ID	27
CTE	28
IF	59
THEN	60
ELSE	61
+	70
/	73
>=	80
=	85

UNIVERSIDAD ANDRES BELLO

Análisis léxico (ejemplo)

[if] [Plazo] [>=] [30]
[then] [Tasa] [=] [Base] [+] [Recargo] [/] [100]
[else] [Tasa] [=] [Base]

Análisis léxico (ejemplo)

[59] [Plazo] [>=] [30]

[then] [Tasa] [=] [Base] [+] [Recargo] [/] [100]

[else] [Tasa] [=] [Base]

Análisis léxico (ejercicio)

Complete la salida del análisis léxico en el programa de ejemplo.

Análisis léxico (solución)

[59] [27, ‘Plazo’] [80] [28, ‘30’]

[60] [27, ‘Tasa’] [85] [27, ‘Base’] [70] [27,’Recargo’] [73] [28, ‘100’]

[61] [27,’Tasa’] [85] [27,‘Base’]

UNIVERSIDAD ANDRES BELLO

Análisis sintáctico

- los tokens son símbolos terminales en la gramática que describe al lenguaje de origen
- la estructura jerárquica de un programa es representada por reglas que constituyen una gramática
- las reglas

Optimización de código

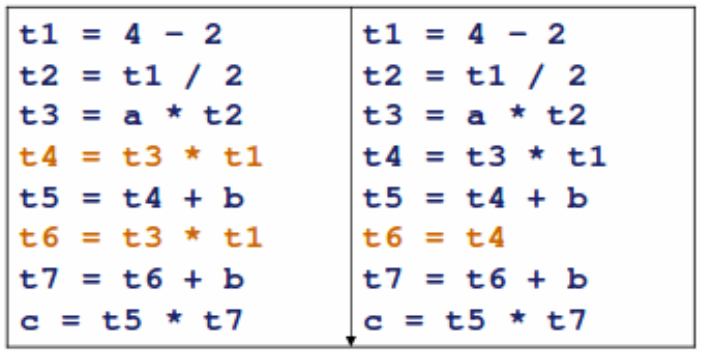
- la optimización consiste en mejorar la calidad y eficiencia del objeto creado para aumentar la velocidad de ejecución y reducir su espacio utilizado
- las optimizaciones de espacio suelen ser incompatibles con las optimizaciones de velocidad (la mejora de una perjudica a la otra)
- las técnicas de mejora se pueden aplicar en un contexto
  - local: solo utiliza información de un bloque básico
  - global: si utiliza información de un conjunto de bloques básicos

Bloques básicos de optimización

- es una unidad fundamental de código, una secuencia de instrucciones en la que el flujo de control entra en el inicio del bloque y sale al final
- en optimización de un bloque básico se debe mantener el valor de las variables a la entrada y a la salida del bloque

Eliminación de subexpresiones comunes

- si una expresión se calcula más de una vez, se puede usar el primer cálculo para reemplazar los cálculos posteriores



```
t1 = 4 - 2
t2 = t1 / 2
t3 = a * t2
t4 = t3 * t1
t5 = t4 + b
t6 = t3 * t1
t7 = t6 + b
c = t5 * t7
```

```
t1 = 4 - 2
t2 = t1 / 2
t3 = a * t2
t4 = t3 * t1
t5 = t4 + b
t6 = t4
t7 = t6 + b
c = t5 * t7
```

UNIVERSIDAD ANDRES BELLO

Eliminación de código muerto

- código que no se ejecutará nunca
- operaciones insustanciales (declaraciones nulas o asignaciones de una variable a sí misma)

`b := false;`
`if (a AND b) then`
`instrucciones`
	`b := false ;`

`int global;`
`void f () {`
`  int i;`
`  i = 1;`
`  global = 1;`
`  global = 2;`
`  return;`
`  global = 3;`
`}`
	`int global;`
	`void f () {`
	`  global = 2;`
	`  return;`
	`}`