



**UNCUYO**  
UNIVERSIDAD  
NACIONAL DE CUYO

# PARADIGMAS DE PROGRAMACIÓN

## UNIDAD 1

Dr. Pablo Vidal

Facultad de Ingeniería  
Universidad Nacional de Cuyo

2023

# Tabla de Contenidos

- 1 Traductores
- 2 Proceso de traducción
- 3 Tipos y ligaduras
- 4 Sistema de Tipados
- 5 Ligaduras
- 6 Sumario

# Traductores

# Traductor



# Traductor



# Proceso de ejecución

```
package main

import "fmt"

func main() {
    fmt.Printf("hello, world")
}
```



```
0101010111101110001101
0100010100010101001010
0101010010101010000101
0011010001010100011110
0110010100101010101001
1110001101010010010001
```

Lenguaje de alto nivel que  
entiende el programador

Lenguaje de máquina que  
entiende el procesador

# Conceptos Básicos

- **Traductor:** Programa que toma como entrada un texto escrito (llamado fuente) y da como salida otro texto (llamado objeto).
- **Compilador:** Traductor cuyo fuente es un lenguaje de alto nivel y cuyo objeto es un lenguaje de bajo nivel.
- **Interprete:** Compilador que ejecuta al mismo tiempo que traduce. Es un programa que no genera un programa equivalente, sino que toma una sentencia del programa fuente en un lenguaje de alto nivel y la traduce al código equivalente y al mismo tiempo lo ejecuta

# Proceso de compilación

```

22
23 function compare_name(sequence a, sequence b)
24 -- Compare two sequences (records) according to NAME.
25 return compare(a.NAME, b.NAME)
26 end function
27
28 function compare_pop(sequence a, sequence b)
29 -- Compare two sequences (records) according to POPULATION.
30 -- Note: comparing b vs. a, rather than a vs. b, makes
31 -- the bigger population case first.
32 return compare(b.POPULATION, a.POPULATION)
33 end function
34
35 sequence sorted_by_pop, sorted_by_name
36 integer by_pop, by_name
37
38 by_pop = routine_id("compare_pop")
39 by_name = routine_id("compare_name")
40
41 sorted_by_pop = custom_sort(by_pop, statistics)
42 sorted_by_name = custom_sort(by_name, statistics)
43
44 parallel, "sorted by population" { sorted_by_name }
45 for i = 1 to length(sorted_by_pop) do
46   print(i, "%20s %10s %10s",
47     sorted_by_pop(i) & sorted_by_name(i))
48 end for
49

```

## Módulos

```

44 parallel, "sorted by population" { sorted_by_name }
45 for i = 1 to length(sorted_by_pop) do
46   print(i, "%20s %10s %10s",
47     sorted_by_pop(i) & sorted_by_name(i))
48 end for
49

```

```

44 parallel, "sorted by population" { sorted_by_name }
45 for i = 1 to length(sorted_by_pop) do
46   print(i, "%20s %10s %10s",
47     sorted_by_pop(i) & sorted_by_name(i))
48 end for
49

```

**Compilación**

**Ejecución**

**Entorno  
(SO)**

**Librerías  
estáticas**

```

C0E8: F8 C8 C9 06 D0 2A A9 00 22
C0F0: 8D F8 C8 8D FD C8 AE 83 C9
C0F8: C1 A9 01 9D 78 C1 A9 80 CA
C100: 8D FF 07 AD 7C 05 8D 81 D2
C108: C1 20 84 C1 AD 20 89 8D 15
C110: F8 89 AD 21 89 8D F9 89 F8

```

```

C0E8: F8 C8 C9 06 D0 2A A9 00 22
C0F0: 8D F8 C8 8D FD C8 AE 83 C9
C0F8: C1 A9 01 9D 78 C1 A9 80 CA
C100: 8D FF 07 AD 7C 05 8D 81 D2
C108: C1 20 84 C1 AD 20 89 8D 15
C110: F8 89 AD 21 89 8D F9 89 F8

```

**Librerías  
dinámicas**

```

C0E8: F8 C8 C9 06 D0 2A A9 00 22
C0F0: 8D F8 C8 8D FD C8 AE 83 C9
C0F8: C1 A9 01 9D 78 C1 A9 80 CA
C100: 8D FF 07 AD 7C 05 8D 81 D2
C108: C1 20 84 C1 AD 20 89 8D 15
C110: F8 89 AD 21 89 8D F9 89 F8

```

```

C0E8: F8 C8 C9 06 D0 2A A9 00 22
C0F0: 8D F8 C8 8D FD C8 AE 83 C9
C0F8: C1 A9 01 9D 78 C1 A9 80 CA
C100: 8D FF 07 AD 7C 05 8D 81 D2
C108: C1 20 84 C1 AD 20 89 8D 15
C110: F8 89 AD 21 89 8D F9 89 F8

```

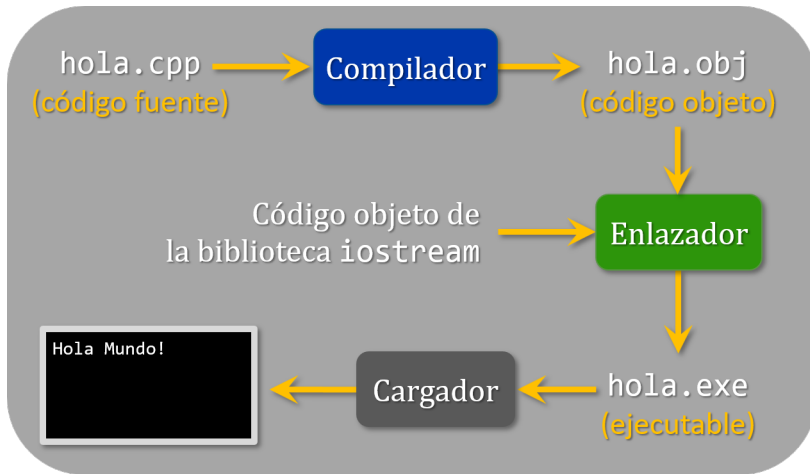


# Ventajas

## Ventajas de los compiladores frente a los interpretes

- Solo se compilan una vez
- Ejecución más rápida
- Optimización de código e información del error más detallada.

# Proceso de compilación - ejemplo





# Interprete

## Programa (Sesión interactiva)

```

22
23 function compare_name(sequence a, sequence b)
24 -- Compare two sequences (records) according to NAME.
25 return compare(a[NAME], b[NAME])
26 end function
27
28 function compare_pop(sequence a, sequence b)
29 -- Compare two sequences (records) according to POPULATION.
30 -- Note: comparing b vs. a, rather than a vs. b, makes
31 -- the bigger population come first.
32 return compare(b[POPULATION], a[POPULATION])
33 end function
34
35 sequence sorted_by_pop, sorted_by_name
36 integer by_pop, by_name
37
38
39
40 sorted_by_pop = custom_sort_by_pop, statistics
41 sorted_by_name = custom_sort_by_name, statistics
42
43
44 puts(1, "sorted by population(1) sorted by name(1)")
45 for i = 1 to length(sorted_by_pop) do
46   print(i, "NOM: %10s POP: %10s",
47     sorted_by_pop[i] & sorted_by_name[i])
48 end for
49

```

Comando actual



Resultado

## Intérprete

```

0000: FB CB C9 06 D0 2A A9 00 22
0001: 8D FB CB 8D FD CB AE E3 C9
0002: C1 A9 01 9D 7B C1 A9 E0 CA
0003: 8D FF 07 AD 7C 05 8D 81 D2
0004: C1 20 84 C1 AD 3F 89 8D 15
0005: FB 89 AD 21 89 8D F9 89 FB

```

```

10 DATA 1C00,A9,4C,8D,05,1E
11 DATA 1C08,02,A0,0B,A2,05
12 DATA 1C10,95,C2,8B,CA,10
13 DATA 1C18,37,1C,EA,EA,EA
14 DATA 1C20,02,E6,CA,E6,C9
15 DATA 1C28,CA,20,A4,CC,20
16 DATA 1C30,00,B1,C9,AS,A6
17 DATA 1C38,CB,00,02,E6,CB

```

Estado  
Sesión

Ejecución

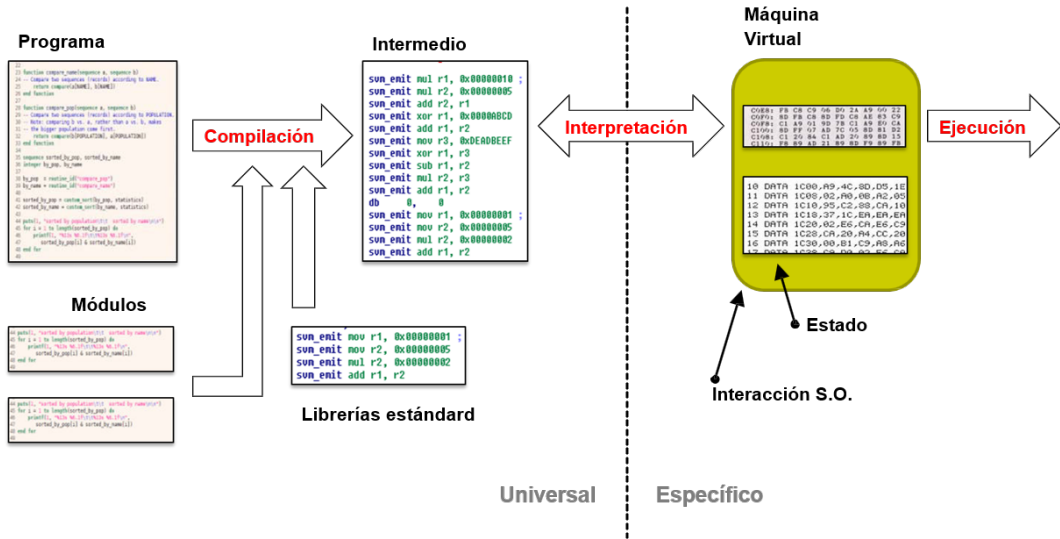
I/O

# Ventajas

## Ventajas de los Interpretes frente a los Compiladores

- Menor coste espacial
- Mayor interactividad en desarrollo
- Añadir código *en caliente*

# Compilación e interpretación



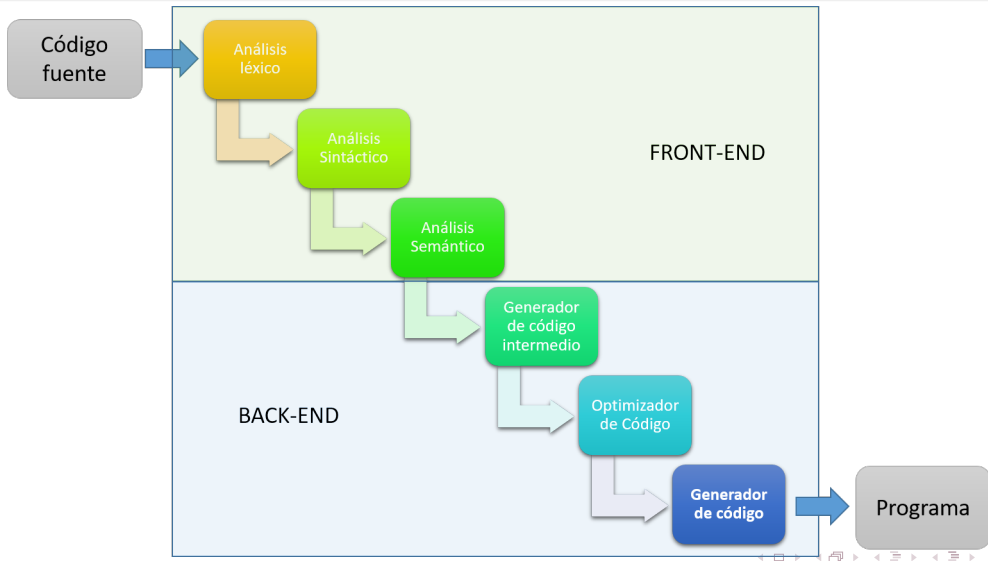
## Proceso de traducción

# Estructura de un compilador

Para la realización del proceso de traducción es necesario dividir el compilador en varias fases.



# Estructura



# Tabla de símbolos

## Estructura

Almacena información creada y mantenida por los compiladores acerca de la ocurrencia de diversas entidades, tales como nombres de variables, funciones, objetos, clases, interfaces, etc.

- Palabras reservadas
- Literales
- Símbolos predefinidos
- Símbolos definidos por el programador
- ...

## El objetivo

- Colaborar con las comprobaciones semánticas.
- Facilitar ayuda a la generación de código

# Análisis léxico

- Lee el programa fuente.
- Remueve espacios en blanco, tabulaciones, saltos de línea.
- Remueve comentarios
- Agrupa lo leído en secuencias significativas conocidas como lexemas. Y para cada lexema este analizado produce como salida un token en la forma  
`{nombre, valor}`

(nombre-token, valor-atributo)

# Análisis léxico

- Palabras reservadas: Ejemplos: IF, THEN, ELSE
- Operadores: Ejemplos: '+', '==', ':='
- Cadenas de múltiples caracteres: Ejemplos: Identificador, Constante

```

int x = 3;
int y = x + 7;
while (x != y) {
    if (x > y) {
        x = x - y;
    } else {
        y = y - x;
    }
}

```



```

("int", KEYWORD)
("x", IDENTIFIER)
("=", OPERATOR)
("3", INT_CONSTANT)
(";", SPECIAL_SYMBOL)
("int", KEYWORD)
("y", IDENTIFIER)
("=", OPERATOR)
("x", IDENTIFIER)
("+", OPERATOR)
("7", INT_CONSTANT)
(";", SPECIAL_SYMBOL)
("while", KEYWORD)
("(" , SPECIAL_SYMBOL)

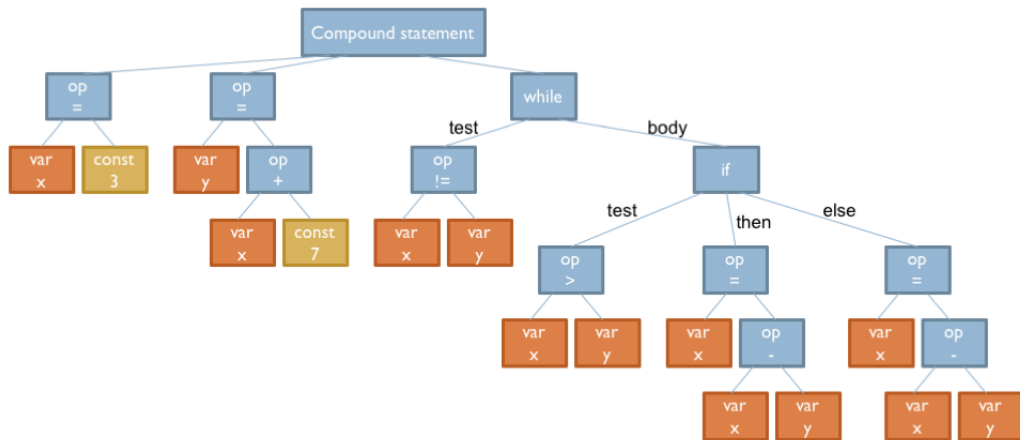
```

...

# Análisis Sintáctico (Parser)

Es la 2da fase de compilación. Utiliza los tokens para crear una representación intermedia en forma de árbol, en la cual cada nodo interior representa una operación y los hijos del nodo representan los argumentos de la operación.





# Análisis Semántico

Utiliza el árbol sintáctico y la información en la tabla de símbolos para comprobar la consistencia semántica del programa fuente con la definición del lenguaje.

Realiza comprobación de tipos, donde verifica que cada operador tenga operandos que coincidan.

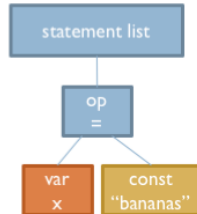


Consider:

```
int x = "bananas";
```

Syntax OK

Semantically (meaning)  
WRONG



Var	Type
x	int

Line 1: error, invalid conversion from string constant to int

# Generador de Código Intermedio

El compilador puede crear 1 o + representaciones intermedias similares al código maquina. La representación debe ser fácil de producir y fácil de traducir en la máquina destino.

# Optimización de código

Trata de mejorar el código intermedio, de manera que produzca un mejor código destino. Mejor significa más rápido, más corto y que consuma menos recursos.

# Generación de Código

- Convierte un programa sintácticamente correcto en una serie de instrucciones a ser interpretadas por una máquina
- La máquina destino puede ser un microprocesador o una máquina abstracta tal como una máquina virtual o un lenguaje intermedio



## COMPILADO



Convierte el código a binarios que lee el sistema operativo.



## INTERPRETADO



Requieren de un programa que lea la instrucción del código en tiempo real, y la ejecute.



## INTERMEDIO



Se compila el código fuente a un lenguaje intermedio y este último se ejecuta en una máquina virtual



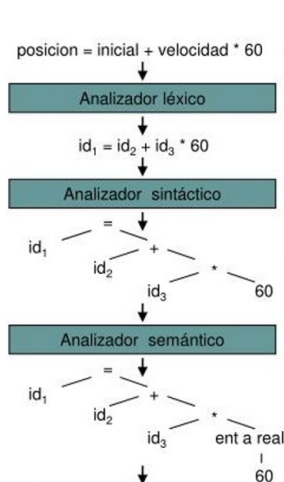
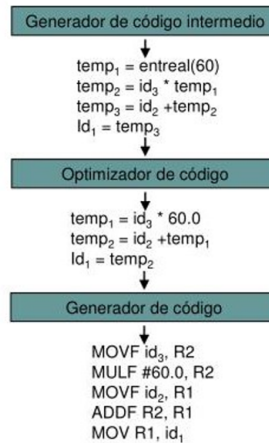


TABLA DE SIMBOLOS

1	posicion	...
2	inicial	...
3	velocidad	...
4		



## Tipos y ligaduras

# Repaso: Valores y variables

- Las variables, los procedimientos o las constantes tienen nombres asignados por el programador
- Los valores asignados son cualquier entidad almacenable como enteros, reales o arreglos.
- Las localizaciones son lugares o zonas de memoria donde se almacenan estos valores.
- El significado de un nombre queda determinado por las propiedades o atributos asociados con el mismo



# Atributos

- Ejemplo: `const int n = 5`
- Esta declaración convierte a `n` en una constante entera con valor 5, es decir, asocia al nombre `n` el atributo de tipo de datos contante entera y el atributo de valor 5

# Variables: atributos

**Variable** (nombre, dirección, tipo, valor, tiempo de vida, ámbito)

# Nombres e identificadores

- Cadena de caracteres utilizado para identificar entradas en un programa
- Consideraciones de diseño:
  - ¿tienen longitud máxima?
  - ¿es sensible a mayúsculas y minúsculas?
  - ¿se permite el uso de conectores?
  - ¿hay palabras especiales (claves o reservadas)?
- Longitud: si es muy restringida afecta la legibilidad del programa. Ej:
  - Fortran: longitud máxima 6
  - Cobol: longitud máxima 30
  - Ada y Java: sin limites, todos los caracteres son significativos
  - C++: sin limites, pero los implementadores generalmente imponen uno
- Sensibilidad a mayúsculas y minúsculas: interfiere seriamente con la facilidad de lectura (readability)

# Atributos de una variable

- **Dirección:** la dirección de memoria a la que está asociada una misma variable puede estar asociada a locaciones de memoria diferentes durante la ejecución del programa, y en diferentes lugares del programa.
- **Aliasing:** dos nombres de variables diferentes pueden utilizarse para acceder a la misma locación de memoria al mismo tiempo
- El aliasing perjudica la legibilidad de los programas, ya que el programador que lee el programa debe tener presente todos los alias de una locación.
- Los alias se pueden crear a través del uso de punteros, variables por referencia.

# Variables: atributos

**Variable:** (**nombre**, **dirección**, **tipo**, valor, tiempo de vida, ámbito)

# Sistema de Tipados

# ¿Qué es un tipo?

- Un tipo describe un conjunto de valores y operaciones predefinidos para esos valores. Al asignarle un tipo a una expresión se está delimitando cuál es el conjunto de valores que podría denotar esa expresión.
- Un sistema de tipos me provee una forma de estudiar qué valores tienen sentido para ser utilizados en una operación dada. Por ejemplo, si  $x$  e  $y$  son valores numéricos, la expresión  $x + y$  tiene sentido. En cambio si  $x$  e  $y$  son valores booleanos,  $x + y$  carece de sentido.
- Toda expresión en un programa puede denotar diferentes valores en diferentes momentos (dependiendo del lenguaje eso puede ocurrir en la misma o en diferentes ejecuciones del programa).

# Tipos

- La idea de tipo nos permite relacionar:
  - un conjunto de valores que tienen ese tipo o son de ese tipo,
  - con las operaciones que pueden ser realizadas sobre esos valores.

Tipo de una variable: determina el rango de valores que puede tomar la variable y el conjunto de operaciones definidas para los valores del tipo.

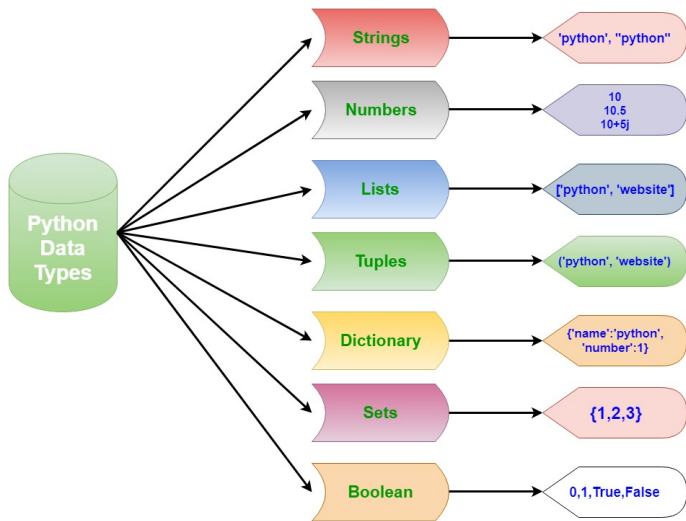
Ej. integer:  $[-32768..32767]$ ,  $+$ ,  $-$ ,  $*$ ,  $/$



# Tipos

- Los sistemas de tipos sirven para hacer programas correctos e impedir que se puedan representar estados incorrectos. *Julie Moroniki - Haskell.*
- Clasifica los valores y las expresiones en tipos, cómo se pueden manipular estos tipos y cómo interactúan

# Ejemplo



# Tipado Estatico y tipado dinamico

- Tipado estático: Un lenguaje de tipo estático es un lenguaje (como Java, C, o C++) en donde los tipos de variables se conocen en tiempo de compilación. En la mayoría de estos lenguajes, los tipos deben ser expresamente indicados por el programador; en otros casos (como en OCaml), la inferencia de tipos permite al programador no indicar sus tipos de variables.

# Tipado Estático y tipado dinámico

- Tipado dinámico: Los lenguajes de tipado dinámico son aquellos donde el intérprete asigna a las variables un tipo durante el tiempo de ejecución basado en su valor en ese momento.

# Tipado fuerte o débil

- Se entiende que un lenguaje tiene un sistema de tipos fuerte cuando un tipo no puede ser cambiado una vez definida la variable nunca. No se podrá operar entre distintos tipos de forma implícita, nosotros como desarrolladores tendremos que hacer las conversiones necesarias de forma explícita.

# Tipado fuerte o debil

- La mayoría de veces, el tipado débil es en donde no indicamos el tipo de variable al declararla. La verdadera diferencia es que podemos asignar, por ejemplo, un valor entero a una variable que anteriormente tenía una cadena.

# Tipado fuerte vs tipado débil

- En general decimos que un lenguaje tiene tipado débil si las reglas sobre lo que se puede hacer con los tipos no son estrictas. Variables de distintos tipos pueden ser mezcladas en distintas operaciones.
- En caso contrario decimos que el lenguaje tiene un tipado fuerte.
- **No debe confundirse nunca con lenguajes dinámicos o estáticos**

# Tipado Nominal y Estructural

- Un sistema de tipado nominal, dos tipos son compatibles si tienen el mismo nombre
- En un sistema de tipado estructural, dos tipos son iguales si uno contiene al menos la misma información que el tipo al que estamos asignando.

## ■ Tipo

- E=entero
- A=vector de 1 a 10 de enteros

## ■ Var

- X1 = entero
- X2 = entero
- X3 = E
- V1 = vector de 1 a 10 de enteros
- V2 = A

## ■ Equivalencia Nominal

- X1 eqN x2
- V2 eqN v3

## ■ Equivalencia Estructural

- X1 eqE X2 eqE X3
- V1 eqE V2 eqE V3



# Ligaduras

# Concepto de Ligadura

- Una ligadura es la asociación entre dos cosas.
- Por ejemplo: se da entre un nombre y la cosa nombrada por él.
- Asociación entre un atributo y una entidad
- **El proceso de asociación de un atributo a una entidad (nombres) se conoce como ligadura (binding)**
- El momento en el que se produce se denomina tiempo de ligadura.

# Declaración, bloques y alcance

- La declaración es el método esencial para establecer ligaduras.
- Las declaraciones están asociadas comúnmente con constructores del lenguaje como los bloques.

# Tiempo de ligadura

- El tiempo de ligadura es el momento en el cual se crea la ligadura o el momento en el cual se toma una decisión de implementación. Hay diferentes momentos a los cuales la decisiones pueden estar vinculadas.
- Dependiendo del momento de asociación la ligadura puede ser **estática**(tiempo de compilación) o **dinámica** (tiempo de ejecución)

# Tiempo de ligadura

- Solo en ejecución se produce lo que se denomina ligadura dinámica, en otros momentos existentes tenemos una ligadura estática.

# Ligadura del tipo a una variable: dinámica

- La correspondencia se hace al asignar valores o bien al invocar los métodos
- Soporta la evolución de los programas sin tener que recompilar
- Coste en tiempo de ejecución
- Se especifica a través de la declaración de asignación:
  - `ls = [2, 4, 6]`, es una lista
  - `ls = 17.3`, es real

## Ventajas y desventajas

- Ventajas: flexibilidad (unidades de programas genéricas)
- Desventajas: Alto costo (Chequeo de tipos dinámicos e interpretación) y detección de errores complicada y en tiempo de ejecución

# Ligaduras



# Alcance de una variable

- Una variable tiene una ligadura activa para una sentencia si puede ser referenciada en ella
- El alcance de una ligadura puede determinarse:
  - Estáticamente
  - Dinámicamente

Regla de alcance: Determinan como referencias de variables declaradas fuera del subprograma en ejecución o bloque son asociadas con sus declaraciones y por lo tanto con sus atributos



# Alcance

- Alcance estático: en un lenguaje con reglas de alcance estático o léxico la ligadura se determina en tiempo de compilación a partir de la examinación del texto del programa.
- Habitualmente la ligadura tiene lugar “matcheando” la declaración cuyo bloque esta mas cercano a ese punto del programa.

# Alcance

Alcance Dinámico: la ligadura se resuelve siguiendo la secuencia de llamados, no en la relación del texto del programa. De ahí que solo pueda determinarse en ejecución.

# Ligaduras

	Comprobación Estática	Comprobación Dinámica
Ligadura Estática	Garantía de corrección Interpretación Inflexible	Combinación no valida (sin sentido)
Ligadura Dinámica	Garantía de corrección Interpretación Flexible	No hay garantía de corrección Interpretación Flexible

# Sumario




# Sumario

En resumen planteamos tres clasificaciones:

- En cuanto a la forma de chequeo puede ser estático/compile time, dinámico/runtime o nada.
- En cuanto a la forma de especificar el tipo de algo puede ser debil o fuerte.
- En cuanto a la forma de constituir un tipo puede ser nominal o estructural.

# Variable

- Valor: es el contenido de la celda o celdas asociadas a la variable.
- Tipo: determina el rango de valores para la variable y las operaciones que están definidas para los valores de ese tipo.
- Ligadura del tipo: antes de que una variable pueda ser utilizada debe estar ligada a un tipo. Es relevante entonces, decidir dos aspectos:
  - 1 Como se especifica el tipo
  - 2 Cuando la ligadura toma lugar

-  ALLEN B. TUCKER, 2007, *Programming Languages, Principles and Paradigms*, cGraw-Hill Higher Education, CAPITULO 5 y 6.
-  AHO, ALFRED, SETHI, RAVI, ULLMAN, JEFFREY D., LAM, MONICA S., 2008, *Compiladores. Principios, técnicas y herramientas*, Addison Wesley.
-  KENNETH C LOUDEN, KENNETH ALFRED LAMBERT, 2011, *Programming languages, principles and practice* Course Technology, Cengage Learning. CAPITULO 8.