



---

MAYO, 2022

# COMPROBACIÓN DE TIPOS COMPILADORES





Un compilador debe comprobar si el programa fuente sigue tanto las convenciones sintácticas como las semánticas del lenguaje fuente. Esta comprobación garantiza la detección y comunicación de algunas clases de errores de programación

Comprobación de tipos – Operador y operandos compatibles

Comprobación de flujo de control – proposiciones que hacen que el flujo de control abandone una construcción

Comprobación de unicidad – Definiciones de objetos solo una vez.

Comprobaciones relacionadas con nombres – Uso de identificadores iguales cuando necesite.



La gestión de tipo debe asegurarse de que la desreferenciación se aplique sólo a un apuntador, de que la indización se haga sólo sobre una matriz, de que una función definida por el usuario se aplique la cantidad y tipo correctos de argumentos, etc.

La sobrecarga puede ir acompañada de una conversión de tipos, donde el compilador proporciona el operador para convertir un operando en el tipo esperado por el contexto. Por ejemplo el símbolo +.

Una función es polimórfica cuando puede ejecutarse con argumentos de tipos diferentes, desencadenando en cada caso una acción diferente. Por ejemplo, la suma es un operador polimórfico, pues permite sumar enteros, reales, reales con enteros, y concatenar cadenas o sumar enteros con caracteres.

## Ejemplo

IMPRIMIR expr.

Esta expr puede ser de dos tipos: entera o bien carácter. Sin embargo la función imprimir es la misma.

Es de tipo POLIMÓRFICA.

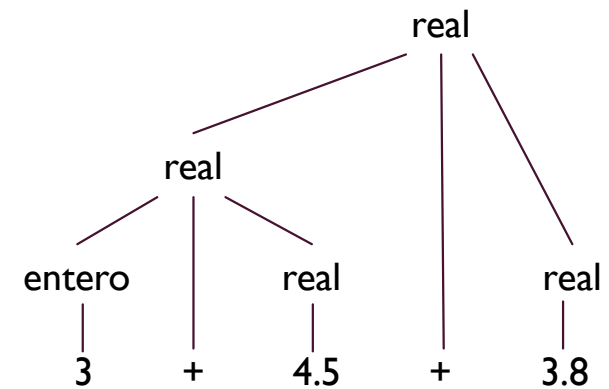
b = 'k'

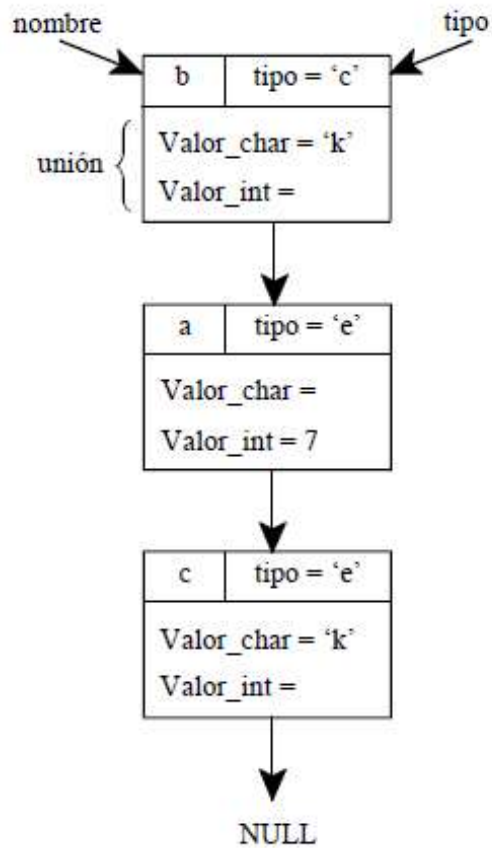
a = 7

c = 12 + 3

IMPRIMIR c + 5

IMPRIMIR b





## Almacenamiento en la tabla de símbolos

$b = 'k'$

$a = 7$

$c = 8 + a$

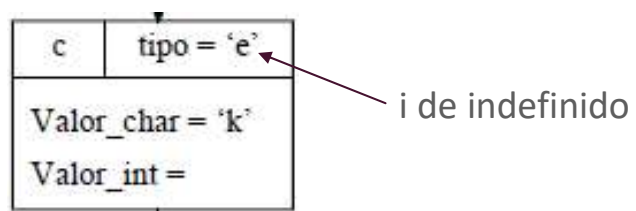
IMPRIMIR  $c + 5$

IMPRIMIR  $b$

¿de qué tipo es la variable  $c$  para insertarlo en la tabla de símbolos?

El tipo está implícito en el momento en que se hace su primera asignación.

¿Qué ocurriría en este caso?  $c = 8 + 'h'$ .



En la función **A\_CHARACTER(expr)** preguntaremos el tipo y:

- Si no es entero nos dará un error
- Si es entero devuelve un carácter

En la función **A\_ENTERO(expr)** preguntaremos el tipo y:

- Si no es carácter nos dará un error
- Si es carácter devuelve un entero

$b = f$

Mete en la tabla de símbolo  $b$  con tipo 'i', se encuentra  $f$  y también la mete con tipo 'i', (El tipo de un identificador no se sabe hasta la primera asignación).

- $b := f$
- $f := 3$

- 
- $b := 3$
  - $b := f$
  - $b := 'k' .$

Cuando hay un error no se le asigna el tipo 'i' porque sino permitiríamos cosas como

$b = 3 \rightarrow$  Funciona

$b = f \rightarrow$  Falla

$b = 'k' \rightarrow$  ¡Funciona! (Pero no debe)

Un tipo indefinido implica un valor desconocido.

Añadir una variable booleana a la tabla de símbolos que me diga si el valor de la variable es conocido o no.

Ver archivo  
TablaSimEjemplo.c



# SISTEMAS DE TIPOS

Los tipos más interesantes son los tipos estructurados o tipos complejos ( array, registros, punteros,...).

*Constructores de tipos* son palabras que extienden el tipo. Por ejemplo

POINTER TO

RECORD OF

ARRAY

PROCEDURE

Estos no son tipos, sino constructores de tipos (Se utilizan en la zona de declaraciones).

Además estos constructores de tipos son recursivos. El problema es que el tipo de una variable puede ser todo lo extenso que queramos, por lo que no podremos definir el tipo con una letra.

# SISTEMAS DE TIPOS

Los *modificadores de tipos* (éstos se utilizan en la parte de sentencias). Por regla general los constructores de tipos tienen sus propios modificadores de tipos. Por ejemplo

Constructores	Modificadores
POINTER TO	^
RECORD OF	.
ARRAY	[]
PROCEDURE	()

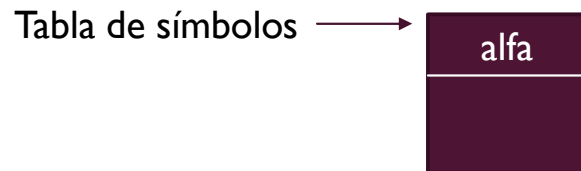
¿Qué vamos a  
permitir en  
nuestro  
lenguaje?

# SISTEMAS DE TIPOS

Supongamos que el usuario define la siguiente variable:

```
alfa : POINTER TO ARRAY [23] OF POINTER TO PROCEDURE ():ARRAY [2] OF INTEGER;
```

En la tabla de símbolos se almacenará el nombre de la variable y el tipo.



El tipo tenemos que almacenarlo de forma inteligente ya que luego nos va a servir para detectar errores, y además permitirá saber qué tipo de modificadores de datos se pueden aplicar:

alfa ^ → Correcto

alfa [2] → ¡Incorrecto!

alfa ( ) → ¡Incorrecto!

# SISTEMAS DE TIPOS

Para simplificar el problema trabajaremos con procedimientos que no tienen parámetros y siempre devuelven un valor. Cada tipo lo vamos a codificar con una letra

POINTER TO → p

ARRAY → a

PROCEDURE → f

BOOLEAN → b

INTEGER → i

REAL → r

CHAR → c

Tabla de símbolos →

alfa
p
a
p
f
a
i

El problema lo solucionaremos con una pila. De forma que

alfa : POINTER TO ARRAY [23] OF POINTER TO PROCEDURE ( ):ARRAY [2] OF INTEGER;

↓  
p

↓  
a

↓  
p

↓  
f

↓  
a

↓  
i

# SISTEMAS DE TIPOS

Ahora es muy fácil saber qué modificador de tipo puedo usar

$\wedge$  si la cima de la pila es “p”

[ ] si la cima de la pila es “a”

( ) si la cima de la pila es “f”

Si  $\alpha$  es un **POINTER TO X**, el tipo de  $\alpha^\wedge$  es X, o lo que es lo mismo, basta con eliminar el tope o cima de la pila para obtener el tipo de X.

Si encontramos una *expr* con un modificador de tipo, hay que controlar que el tipo principal de tal expresión, coincide con el modificador de tipo.

Por ejemplo, si  $\alpha^\wedge$ ,  $\alpha[\text{num}]$  ó  $\alpha( )$  hay que controlar que el tipo principal de  $\alpha$  sea *p*, *a* ó *f* respectivamente.

# SISTEMAS DE TIPOS

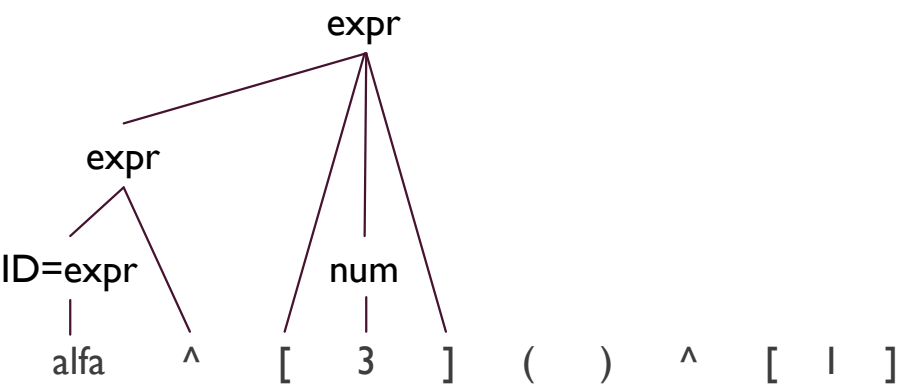
En cualquier caso, necesitamos una gramática para poder manejar toda esas expresiones.

expr : ID

| expr 'Λ'

```
| expr '(' ')
```

```
| expr '[' NUM ']'
```



# SISTEMAS DE TIPOS

La variable *alfa* podemos verla como un *ID* o bien como una *expr*, sea como sea, el tipo de *alfa* es el mismo.

Como *ID* tiene un atributo que es un puntero a la tabla de símbolos, donde se guarda el tipo, mientras que como *expr* tiene como atributo una pila.

alfa ^ [ 3 ] ( ) ^ [ I ]

alfa
p
a
p
f
a
i

Los constructores meten en la pila, y los modificadores sacan de la pila.  
Es decir, a medida que se encuentran modificadores de tipos se destruye, y conforme se encuentran constructores de tipos se construye la pila.

alfa ^ [ 3 ] ( ) ^ [ I ]

alfa
a
p
f
a
i



Si se encuentra un modificador que no es el que se esperaba, entonces se emite un error y en la pila pondremos *u* (undefined). (No podemos usar la *i* de “indefinido” porque se confunde con la *i* de “integer”)