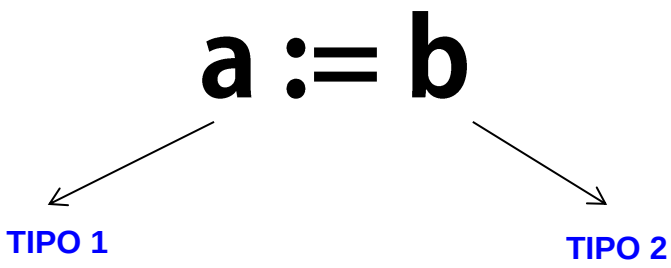
Lenguajes y Compiladores

Compatibilidad y Conversiones

Compatibilidad



Si el lenguaje es dinámico ⇒ **el *tipo2* destruye al *tipo1* y la variable *a* cambia de tipo**

Si el lenguaje orientado a pila (tipo Algol) ⇒ **el *tipo2* se convierte en *tipo1***

Tomemos los lenguajes tipo algol donde el lado izquierdo es el que manda.

Si ***a := b*** se ejecuta, entonces ***tipo1 y tipo2*** son **compatibles** y el lenguaje realiza una **conversión implícita** de ***tipo2*** a ***tipo1***.

Compatibilidad por estructura

Ejemplo en el lenguaje C

```
typedef int unint;
typedef int otroint ;

unint un;
otroint otro;

un = 30 ;
otro = 40 ;
un = otro ;  // Ok compatibilidad por estructura
```

Esta asignación está permitida en el lenguaje C Standard

para los dos tipos existe la *misma representación o estructura interna*.

COMPATIBILIDAD POR ESTRUCTURA

Compatibilidad por estructura

Ejemplo en ADA:

type pesos is new float
type dolar is new float

a,b: pesos
c,d: dolar
e,f: float

a = c

Error de compilación!

Las variables son incompatibles : *a* es de *tipo* *pesos*
c es de *tipo* *dólar*

Las variables poseen la misma estructura → **Las variables tienen distinto nombre de tipo**

COMPATIBILIDAD POR NOMBRE

Compatibilidad por estructura

¿Cómo se soluciona el error de compilación?

Realizando una conversión explícita

type pesos is new float
type dolar is new float

a,b: pesos
c,d: dolar
e,f: float

~~***a = c***~~

a = pesos (c)

Compatibilidad y conversiones

“COMPATIBILIDAD” y “CONVERSIONES”, no resultan temas aislados

Las conversiones las debe realizar el lenguaje o el usuario para que dos tipos en una asignación sean compatibles.

Conversiones

¿Qué es una conversión?  Un cambio de tipos

¿Para qué existe una conversión?  Para permitir la compatibilidad

Existen dos tipos de conversiones

Explícita: el lenguaje exige que el usuario la escriba con claridad.

Implícita: el lenguaje la realiza de forma automática.

Conversiones

Conversión Explícita

Conversiones explicitas

Las conversiones explícitas

- dificultan la escritura
- facilitan la lectura y hacen el código mas legible
- facilitan el mantenimiento

En general va acompañada por la ***compatibilidad por nombre***

Lenguajes : C++. Ada , Java (cast) y otros

Conversiones explicitas

Ejemplo en ADA

```
type PERAS is NEW FLOAT;
type NARANJAS is NEW FLOAT;

A:PERAS;
B:NARANJAS;
C:FLOAT;
C:=A+B                          // No se pueden sumar PERAS con NARANJAS
                                //  por lo tanto hay error de compilación

C:=A+PERAS(B);                  // C espera un float y recibe PERAS
C:=NARANJAS(A)+B;               // C espera un float y recibe NARANJAS
C:=FLOAT(A+PERAS(B));           // Correcto
C:=FLOAT(NARANJAS(A)+B));       // Correcto
C:=FLOAT(A)+FLOAT(B);           // Correcto
```

Conversiones explicitas

Ejemplo en Java

```
byte a = 10;
byte b = 20;
int c = a * b;   // c espera un int  y recibe int (convirtió a *b en int)
```

```
byte a = 15;
a = a*21;          // "Explicit cast needed to convert int to byte"
                   // a espera un byte y recibe un int (a*21 es int)
```

```
a = a*byte(21);          // Correcto
```

Conversiones explicitas

Ejemplo en C# (UNBOXING)

Unboxing es una conversión explícita de un tipo objeto (referencia) a un tipo valor (tipo standard). Siempre que existe un unboxing, hubo un boxing.

```
int ejValor =10;          // variable ejValor de tipo entero

object  ejobj = 200;  //object es un alias de la clase System.object
                      // las vbles de tipo object admiten cualquier valor
                      // boxing

ejValor = (int) ejobj;  // conversión explícita para un tipo referencia a un tipo valor
                        // unboxing
```

Conversiones

Conversión Implícita

Conversiones implícitas

Las conversiones implícitas

- dificultan la lectura
- facilitan la escritura
- dificultan la legibilidad y el mantenimiento

En general va acompañada por la **compatibilidad por estructura**.
Lenguajes C, Algol, Pascal, Fortran, Java

Conversiones implícitas

El compilador realiza todo el “esfuerzo”, de acuerdo con la expresión y el tipo de variable involucrada. Existen reglas para este tipo de conversiones (llamadas generalmente promociones numéricas)

En C

short int → *int* → *long int* → ***double***
unsigned short int →*insigned int* → *unsigned long int* → ***double***
float → ***double***

Tamaños:

short int / int	***2 bytes***
long / float	***4 bytes***
double	***8 bytes***

En Java

byte → *short int* → *int* → *long int* → ***double***
float → ***double***

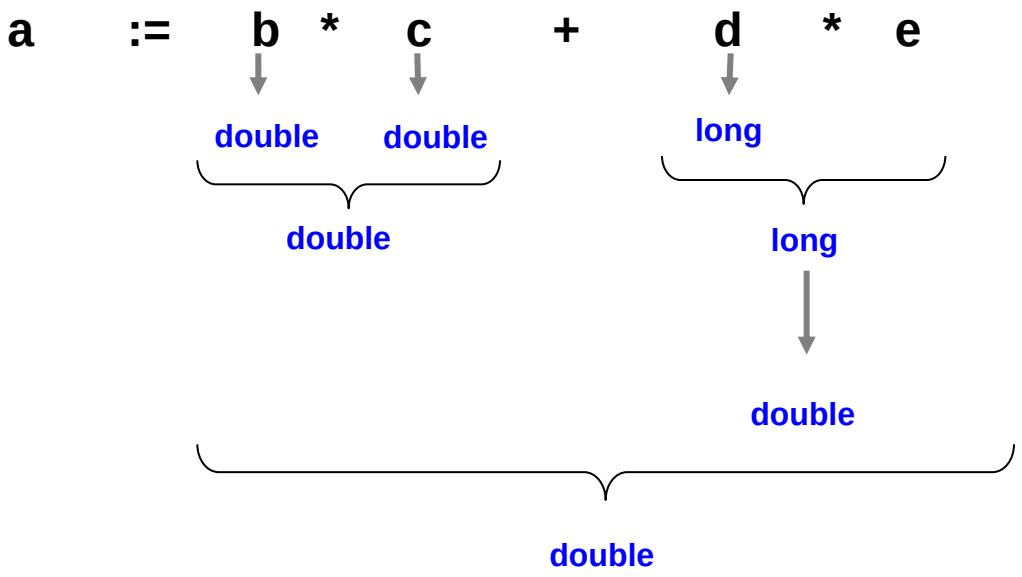
Tamaños:

byte	***1 byte***
short int	***2 bytes***
int / float	***4 bytes***
long	***8 bytes***
double	***8 bytes***

Conversiones implícitas

Ejemplo en C

```
double a
float  c
int d
long e,b
```




```
FILD W b        ; carga b (long) al copro, pasando a double
FMUL DW c       ; multiplica ambos en el copro, porque son de 4 bytes
MOV ECX d       ; como d es int usa el registro extendido para pasar a long
IMUL ECX e      ; multiplica por e (long), obteniendo el resultado en ECX
MOV aux ECX     ; mueve el resultado de la multiplicación (d*e) a una variable para subir al copro
FILD W aux      ; sube la variable aux con el producto al copro (no permite subir registros)
FADD            ; suma ambos en la pila (en la posición 1)
FSTP DW a       ; baja a la variable a, el tope de la pila
```

Conversiones implícitas

Coerciones en Algol

El lenguaje Algol es el lenguaje con más conversiones implícitas en la historia. Estas conversiones no solamente son numéricas

.

¿Todos los lenguajes las heredaron ?

No completamente. Algunas de estas conversiones fueron heredadas por algunos lenguajes.

Conversiones implícitas

Widening (ensanchamiento)

Un valor de un determinado tipo, puede ser asignado a una variable de un tipo más amplio

```
real x;
int z;
x:=z;      // Widening o ensanchamiento del lado derecho (entero) al lado
           // izquierdo (real)
```

Esto prosperó en muchos los lenguajes

Conversiones implícitas

Voiding

Históricamente, Algol introdujo el concepto de void: Un valor puede ser asignado a una variable nula, perdiéndose en este caso el valor asignado.

```
int MM (int a, int b);
int x,y;
void a;          //el tipo no ocupa lugar, se utiliza para ignorar el resultado
a:=MM(x,y);      // Voiding, el lado izquierdo es void y el derecho es una
                    función que devuelve un entero que no será utilizado.
```

Esto prosperó en C, Java , C#, etc. para los «tipos» void.

```
void hola(void) {
                printf("Hola Mundo!\n");  En Java                }
```

"VOID" **no es un tipo en sí, no ocupa lugar.**
El llamado ***"voiding"*** **es una alternativa para descartar el resultado de un procedimiento al asignárselo a una variable.**

```
real c;
void b := c      // (asignación sin sentido)
```

Conversiones implícitas

Voiding en C#

En C#, entre otras cosas, la palabra clave void se utiliza para especificar que un método no devuelve ningun valor. No existen variables void
Aunque se habla de tipos, void no es un tipo.

```
using System;

class Voiding
{
   static void Ejemplo()
   {
      Console.WriteLine("Esto es un ejemplo de voiding");
   }

   static void Main()
   {

       Ejemplo();       // Correcto

      // int var = Ejemplo(); // Error de compilación "Cannot implicitly convert type
                                                                    void to int"


   }
}
```

Conversiones implícitas

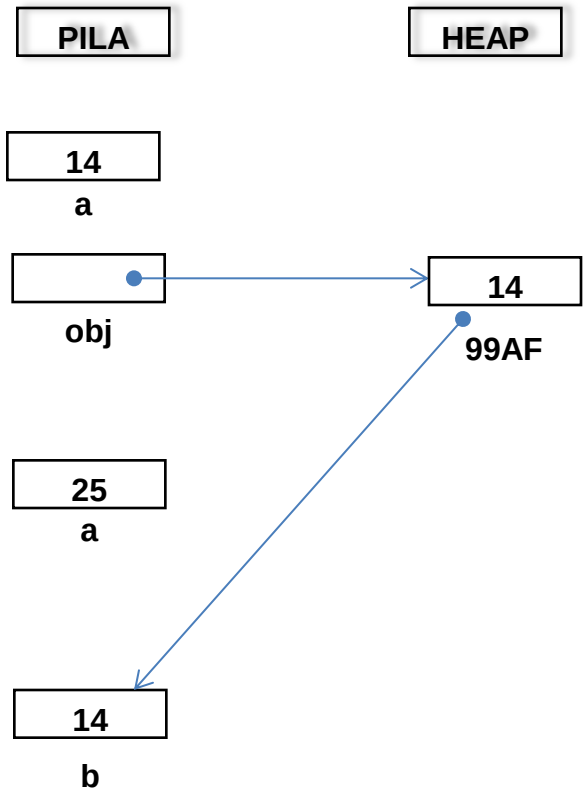
Boxing y Unboxing

Boxing → Convierte un tipo valor a un tipo objeto
Es una conversión implícita

Unboxing → Convierte un objeto a un tipo valor
Es una conversión explícita

```
int a = 14;
object obj = a;
```

La conversión de **boxing** convierte una variable int a un objeto y consiste en crear una variable dinámica en el heap, y asignarle el valor de *a*



int a = 25; // no cambia el valor de la variable referenciada por obj.

int b = (int)obj; // explicita y del mismo tipo que b

La conversión de unboxing convierte una variable objeto a una variable de tipo valor y consiste en llegar a la variable referenciada por obj y copiar su valor en b. ***NO ES POSIBLE QUE HAYA UN UNBOXING SI ANTES NO HUBO UN BOXING***

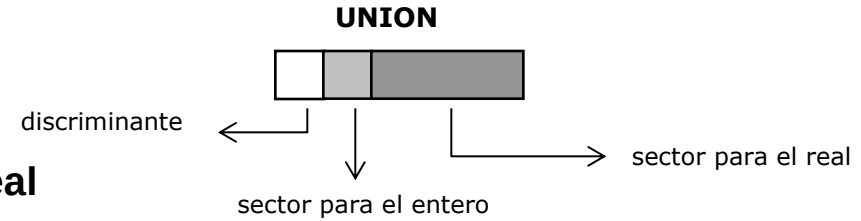
Conversiones implícitas

Uniting

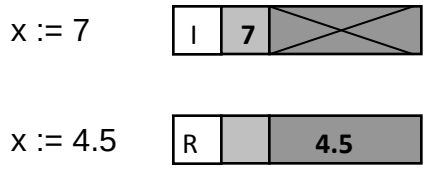
Una variable tipo unión es aquella que posee dos o más datos que ocupan el mismo lugar en diferente momento, es decir, reserva el espacio más grande que el más grande de los tipos de datos que la conforman. Las uniones son útiles para hacer más flexible el código, ahorrar espacio y por consecuencia memoria y finalmente dar soporte de tipos dinámicos a lenguajes tipo Algol.

MODE nuevaunion UNION (integer,real);

x : nuevaunion // x puede cambiar a integer ó real



Para determinar cual es el tipo de valor almacenado por una unión se utiliza el ***discriminante***. Este está conformado por algunos bytes pegados a la variable, que indican si el valor a almacenar es entero o real.
El discriminante funciona como un descriptor



Cuando el lado derecho es de un tipo standard y el izquierdo es un tipo unión, se realiza la asignación y se produce la conversión (uniting).

Conversiones implícitas

Uniting

¿Cómo prosperaron las uniones en otros lenguajes?

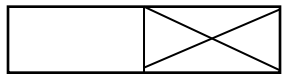
Las variables tipo unión tienen siempre las mismas características pero difieren sus formatos para los distintos lenguajes.

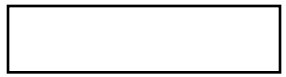
En el lenguaje C

- son llamadas ***uniones*** y se expresan mediante la cláusula ***union***.
- no poseen discriminante y son *uniones inseguras*.
- se escribe un discriminante o no según el programador desee.

```
union u {
   int    i;
   float  f;
  } b;
```

En este ejemplo, el compilador reserva 4 bytes (2 bytes para *int* y 4 bytes para *real*)

Si hay un *int* desperdicia una parte →  ***b.i:=16***

Si hay un *real* se utiliza todo →  ***b.f:=0.16***

Conversiones implícitas


```
int main(void)
{
  struct uno {
    int     i;
    char*    f;
   } a;

   union dos {
    int     i;
    char*   f;
   } b,c;

   union tres {
    int     i;
    char*   f;
   } d;

   a.i = 4000 ;
   a.f = "aaaaaaaaaaaaaaaaaaaaa" ; // no
pierdo el valor de a.i

   b.i = 5000 ;
   b.f = "bbbbbbbbbbbbbbbbbbbb" ; // se
pierde el valor de b.i

// d = b ; // Error, compatibilidad por
                nombre para las uniones
  return 0;
}
```

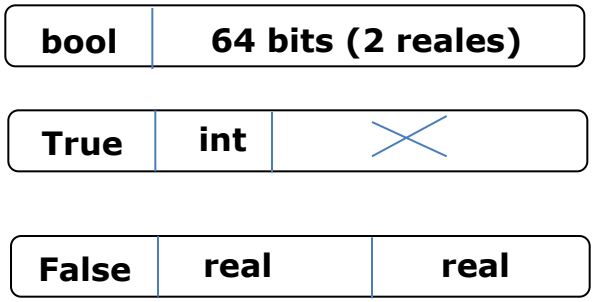
Conversiones implícitas

Uniting

En el lenguaje Pascal

- son llamadas ***registros variantes.***
- poseen discriminantes explícitos.
- *son inseguras*

```
program regvariante
type
producto = record


case stock: boolean  of

            true: (cant : integer);

            false: (ordenado: real, esperado: real)

        end;
var
   p1 : producto;
begin
p1.stock := true;
p1.cant := 10;
p1.ordenado:=2.73;

end.
```



parte variante con un discriminante llamado stock

Permitido

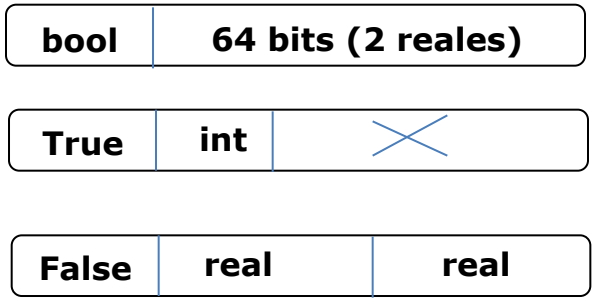
La unión no es segura

Conversiones implícitas

Uniting

En el lenguaje Ada

- son llamadas ***registros variantes.***
- poseen discriminantes explícitos.
- *son seguras*




```
program regvariante

typeprocedure reg_variante is

type producto (stock: boolean :=true) is    –-inicialización necesaria -para uniones
        record
                case stock is
                    when true =>cant : integer;
                    when false =>ordenado : float; esperado : float;
                end case;
        end record;
a  : producto;                      -- registro libre
r,t: producto (stock => false);     -- registro congelado
begin

a:= (stock => true, cant => 10);
r:= (stock => false, ordenado => 2.73, esperado => 3.5);
t:= (stock => false, ordenado => 2.73 );        -- Error Not value for  component esperado

end reg_variante;
```

Ada no permite faltantes en la asignación de una unión

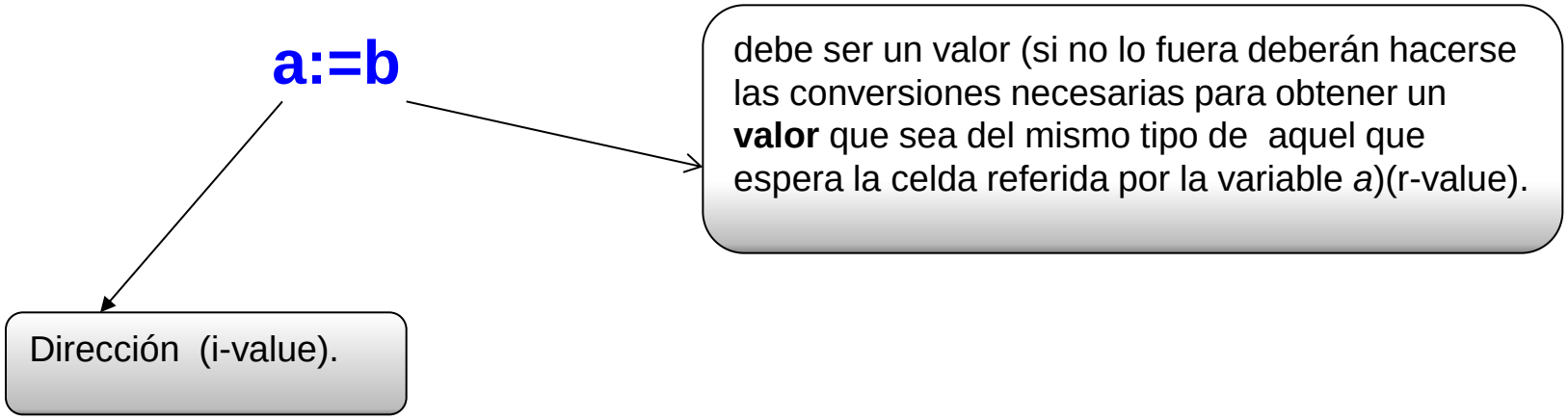
Conversiones implícitas

LENGUAJE	NOMBRE	DISCRIMINANTE	SEGURIDAD
Pascal	Registros Variantes	Explicito (Con Nombre)	Inseguras
C	Uniones	No Tiene	Inseguras
Ada	Registros Variantes	Explicito (Con Nombre)	Seguras

Conversiones implícitas

Dereferencing

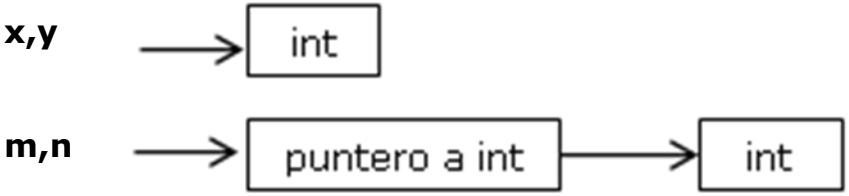
- Algol fue el primer lenguaje que hace uso de punteros
- Se intentó distinguir la noción de dirección y de contenido con respecto a las variables.
- En Algol el concepto era “una variable es la dirección de una celda”.



Conversiones implícitas

Dereferencing

Entonces, según Algol, las variables son referencias a celdas, por lo tanto hay que convertir el lado derecho (desreferenciar), es decir, obtener el valor necesario que necesita la estructura de datos a la que se refiere el i-value.
El dereferencing es la principal operación sobre punteros.



x:=y ; copia un entero **(1 dereferencing)**

x:=m; copia un entero **(2 dereferencing)**

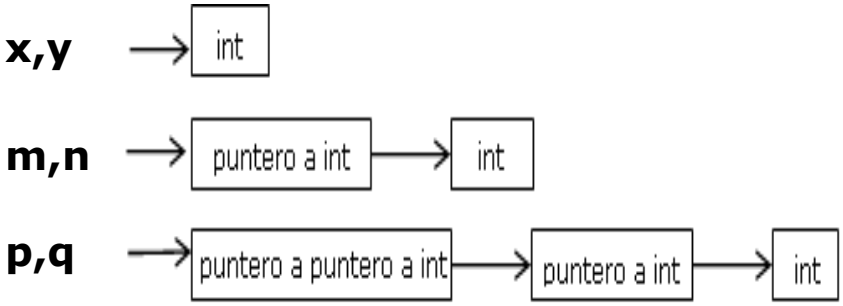
m:=x; copia la dirección de un entero **(0 dereferencing)**

m:=n; copia la dirección de un entero **(1 dereferencing)**

En Algol cuando el lado derecho posee un valor que no satisface las exigencias del lado izquierdo, entonces se realizan tantas extracciones de valor como sean necesarias. **Es totalmente implícito**.

Conversiones implícitas

Dereferencing en LENGUAJE C




```
int x,y;

int *m; int *n;

int ** p ; int **q
```

x:=y ; copia un entero(1 dereferencing)
x:=*m; copia un entero(2 dereferencing)
x:=p;** copia un entero(3 dereferencing)

m:=&x; copia la dirección de un entero(0 dereferencing)
m:=n; copia la dirección de un entero(1 dereferencing)
m:=*p; copia la dirección de un entero(2 dereferencing)

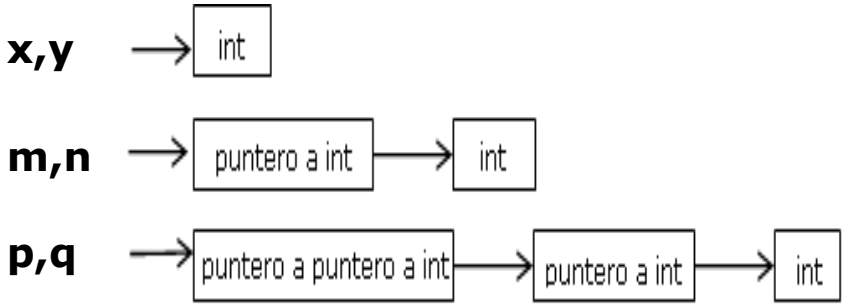
p:=x; copia la dirección de la dirección de un entero(error)
p:=&n; copia la dirección de la dirección de un entero(0 dereferencing)
p:=q; copia la dirección de la dirección de un entero(1 dereferencing)

En C existe una extracción de valor implícita fija, por lo que si se necesitan más hay que escribirlas. Cuando no hay **dereferencing** en el lenguaje C se utiliza *&* que se suele denominar ***supresor de dereferencing*.**

Conversiones implícitas

Dereferencing

Comparando Algol y C

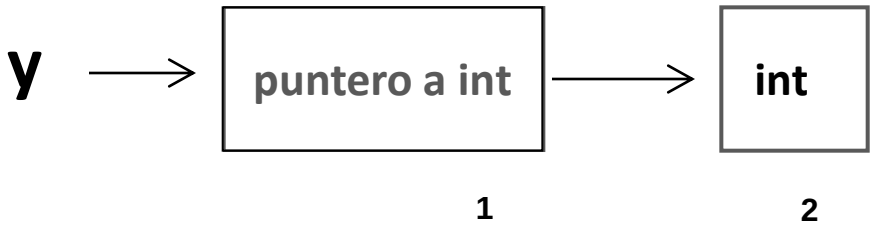


C		ALGOL
x:=y ;	(1 dereferencing)	x:=y ;
x:=*m;	(2 dereferencing)	x:=m;
m:=&x;	(0 dereferencing)	m:=x;
m:=n;	(1 dereferencing)	m:=n;

Conversiones implícitas

Dereferencing

Comparando Algol y C



¿Qué pasaría si se quisiera colocar el valor entero 4 en la celda numerada dos?

EN ALGOL	EN C

ref int y; “declaración"
 	
(ref int)y:=4;
	int * y; // declaración
	
	*y:=4;

Comparando Python y C

PYTHON

```
a = 5
b = a
print "a = {0:d}; b = {1:d}".format(a,b)          # (1)

a = 6
print "a = {0:d}; b = {1:d}".format(a,b)          # (2)
```

(1): a = 5; b = 5

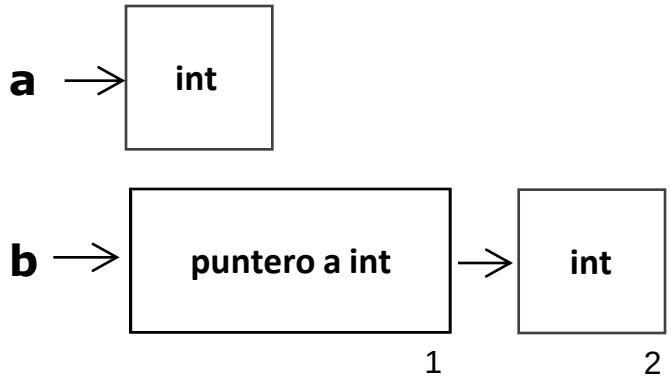
(2): a = 6; b = 5

C

```
#include <stdio.h>
 int main(void)
 int a; a = 5
 int *b; b = &a
 printf("a = %d; b = %d\n", a, *b); // (1)
a = 6;
printf("a = %d; b = %d\n", a, *b); // (2) return 0; }
```

(1): a = 5; b = 5

(2): a = 6; b = 6



Conversiones

¿Preguntas?