

# TIPOS DE DATOS

## CONCEPTOS Y PARADIGMAS DE LENGUAJES DE PROGRAMACIÓN

### TIPOS DE DATOS

# TIPOS DE DATOS

- CONCEPTO
- HISTORIA
- TIPOS DE DATOS
- TIPOS PREDEFINIDOS
- TIPOS DEFINIDOS POR EL USUARIO
- CONSTRUCTORES
- PUNTEROS Y MANEJO DE MEMORIA
- TADs
- SISTEMA DE TIPOS

# TIPOS DE DATOS

CONCEPTO

# TIPOS DE DATOS - CONCEPTO

## Lenguajes de Programación

- Organización de los datos a través del concepto de tipo.
- Forma de clasificar datos de acuerdo con diferentes categorías.
- Más que un conjunto de datos pues tienen un comportamiento semántico o sentido.

Podemos definir a un tipo como un conjunto de valores y un conjunto de operaciones que se pueden utilizar para manipularlos.

# TIPOS DE DATOS

HISTORIA

# TIPOS DE DATOS - HISTORIA

- Los primeros lenguajes tenían el inconveniente de que las estructuras de datos podían ser modeladas sólo con los pocos tipos de datos básicos definidos por el lenguaje.
- Se empieza a ver una clara intención de soportar varios y distintos tipos de datos con el objeto brindar un mayor apoyo al desarrollo de una amplia variedad de aplicaciones. (Legibilidad – Modificabilidad)
- Tomando el concepto de tipo de dato definido por el usuario arribamos al concepto de **tipo de dato abstracto**. Separa la representación y conjunto de operaciones (invisibles al usuario).
- La evolución del TAD es el concepto de Clase.

# TIPOS DE DATOS

TIPO DE DATO

# TIPOS DE DATOS – TIPO DE DATO

	ELEMENTALES	COMPUESTOS
PREDEFINIDOS	ENTEROS	STRING
	REALES	
	CARACTERES	
	BOOLEANOS	
DEFINIDOS POR EL USUARIO	ENUMERADOS	ARREGLOS
		REGISTROS
		LISTAS
		ETC.
DOMINIO DE UN TIPO ----> VALORES POSIBLES		



# TIPOS DE DATOS – TIPO DE DATO

Cualquier lenguaje de programación está **equipado con** un conjunto finito de **tipos predefinidos** ( built-in / primitivos ), que normalmente **reflejan el comportamiento del hardware subyacente**. A nivel de hardware, los valores pertenecen al dominio sin tipo, lo que constituye el dominio universal, estos; son interpretados de manera diferente, según los diferentes tipos que se utilicen.

Los Lenguajes de programación permiten al programador **especificar agrupaciones de objetos de datos elementales** (o tipos predefinidos) y, de forma recursiva, agregaciones de agregados. Esto se logra **mediante** la prestación de una serie de **constructores** que permiten definir lo que denominamos **tipo de dato definido por el usuario**.

# TIPOS DE DATOS

## TIPOS PREDEFINIDOS

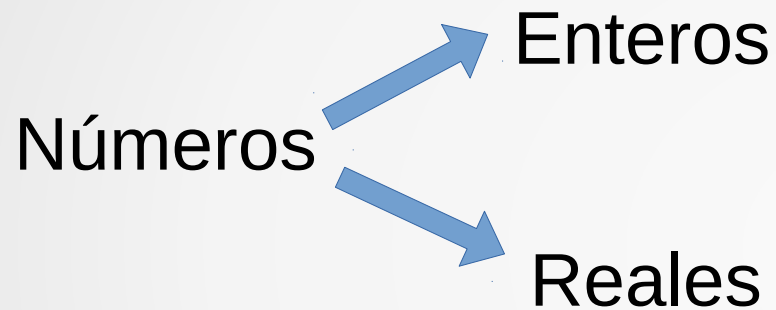
# TIPOS DE DATOS – TIPOS PREDEFINIDOS

## TIPOS PREDEFINIDOS

- Reflejan el comportamiento del hardware subyacente y son una abstracción de él.
- Las ventajas de los tipos predefinidos son:
  - Invisibilidad de la representación
  - Verificación estática
  - Desambiguar operadores
  - Control de precisión

# TIPOS DE DATOS – TIPOS PREDEFINIDOS

## TIPOS PREDEFINIDOS



Caracteres

Booleano

Valores

Longitud Máxima

Conversiones

**Que un conjunto de valores de un tipo sea definido por la implementación del lenguaje significa que será seleccionado por el compilador, mientras que si el tipo es definido por el lenguaje será definido en su definición.**

# TIPOS DE DATOS

TIPOS DEFINIDOS POR EL USUARIO

# TIPOS DE DATOS – DEFINIDOS POR EL USUARIO

## TIPOS DEFINIDOS POR EL USUARIO

Los Lenguajes de programación permiten al programador **especificar agrupaciones de objetos de datos elementales** (o tipos predefinidos) y, de forma recursiva, agregaciones de agregados. Esto se logra **mediante** la prestación de una serie de **constructores** que permiten definir lo que denominamos **tipo de dato definido por el usuario**.

**Separan la especificación de la implementación. Se definen los tipos que el problema necesita.**

- Definir nuevos tipos e instanciarlos
- Chequeo de consistencia

# TIPOS DE DATOS – DEFINIDOS POR EL USUARIO

## Ventajas

- Legibilidad : elección apropiada de nuevos Nombres  
dias [0..31]  
vec [dias] vec[0..31]
- Estructura jerárquica de las definiciones de tipos: proceso de refinamiento  
Record Persona {  
String nombre,apellido;  
int edad;  
.....  
}  
Persona vecino= new Persona(...);  
vecinos = array[1..10] of Persona;
- Modificabilidad : Solo se cambia en la definición
- Factorización: Se usa la cantidad de veces necesarias
- La instanciación de los objetos en un tipo dado implica una descripción abstracta de sus valores. Los detalles de la implementación solo quedan en la definición del tipo

# TIPOS DE DATOS – DEFINIDOS POR EL USUARIO

## COMPUESTOS – CONSTRUCTORES

- Producto Cartesiano
- Correspondencia Finita
- Unión
- Recursión



# TIPOS DE DATOS – DEFINIDOS POR EL USUARIO

## Producto Cartesiano

El producto cartesiano de  $n$  conjuntos  $A_1, A_2, \dots, A_n$ , denotado  $A_1 \times A_2 \times \dots \times A_n$ , es un conjunto cuyos elementos están ordenados  $n$ -tuplas  $(a_1, a_2, \dots, a_n)$ , donde cada  $a_k$  pertenece a  $A_k$ . Por ejemplo, un polígono puede ser descritos por un número entero (sus lados) y un real (longitud de cada borde).

De esta forma el polígono sería un elemento del conjunto del producto cartesiano entre el conjunto de los enteros y el conjunto de los reales.

Estructura de datos?????-----> Estructuras / Registro

# TIPOS DE DATOS – DEFINIDOS POR EL USUARIO

## Producto Cartesiano

- C: estructuras

```
typedef struct {  
    int nro_lados;  
    float tamaño_lado;  
} reg_poligon;
```

Definición

Campos

```
reg_poligon pol = {3, 3.45};  
pol.nro_lados = 4;
```

Instancia con valor  
compuesto inicial

Acceso mediante  
notación puntual

# TIPOS DE DATOS – DEFINIDOS POR EL USUARIO

## Correspondencia Finita

Correspondencia finita es una función de un conjunto finito de valores de un tipo de dominio DT en valores de un tipo del dominio RT.

correspondencia finita en general

$$f: DT \longrightarrow RT$$

Si DT es un subrango de enteros

$$f: [li..ls] \longrightarrow RT$$

**conjunto de valores accesibles via un subindice**

# TIPOS DE DATOS – DEFINIDOS POR EL USUARIO

## Correspondencia Finita

- C (arreglos)

`char digits [10];` ← Definición

`[0..9]` ← Rango del índice / Tipo del dominio

`char` ← Tipo del resultado

`for (i=0; i < 10; ++i)`

`digits[i] = ";` ← Acceso

Phyton: `l = ["una lista", [1, 2]]`

`mi_var = l[1][0]` # `mi_var` vale 1

# TIPOS DE DATOS – DEFINIDOS POR EL USUARIO

## Unión y Unión Discriminada

La unión / union discriminada de dos o mas tipos define un tipo como la disjunción de los tipos dados.

- Permite manipular diferentes tipos en distinto momento de la ejecución
- Chequeo dinámico

# TIPOS DE DATOS – DEFINIDOS POR EL USUARIO

## Unión y Unión Discriminada

La declaración es muy similar a la del producto cartesiano. La diferencia es que sus campos son mutuamente excluyentes.

```
union address{//campos mutuamente exclusivos  
short int offset;  
long unsigned int absoluto;  
};
```

# TIPOS DE DATOS – DEFINIDOS POR EL USUARIO

## Unión y Unión Discriminada

Los valores del tipo `address` deben ser tratados en forma distinta si son de un tipo o del otro.

```
enum descriptor {abs, off}
```

En manos del programador

```
typedef struct {  
  address location;  
  descriptor kind;  
}address_seguro;
```

# TIPOS DE DATOS – DEFINIDOS POR EL USUARIO

## Unión y Unión Discriminada

Unión discriminada agrega un discriminante para indicar la opción elegida.

Si tenemos la unión discriminada de dos conjuntos  $S$  y  $T$ , y aplicamos el discriminante a un elemento  $e$  perteneciente a la unión discriminada devolverá  $S$  o  $T$ .

- El elemento  $e$  debe manipularse de acuerdo al valor del discriminante.
- En la unión y en la unión discriminada el chequeo de tipos debe hacerse en ejecución.
- La unión discriminada se puede manejar en forma segura consultando el discriminante antes de utilizar el valor del elemento.



# TIPOS DE DATOS – DEFINIDOS POR EL USUARIO

## Unión y Unión Discriminada

### Problemas:

- El discriminante y las variantes pueden manejarse independientemente uno de otros.
- La implementación del lenguaje puede ignorar los chequeos.
- Puede omitirse el discriminante, con lo cual aunque se quisiera no se puede chequear.

# TIPOS DE DATOS – DEFINIDOS POR EL USUARIO

## Recursión

Un tipo de dato recursivo  $T$  se define como una estructura que puede contener componentes del tipo  $T$ .

- Define datos agrupados:
  - cuyo tamaño puede crecer arbitrariamente
  - cuya estructura puede ser arbitrariamente compleja.

# TIPOS DE DATOS – DEFINIDOS POR EL USUARIO

## Recursión

Los lenguajes de programación convencionales soportan la implementación de los tipos de datos recursivos a través de los punteros.

- Un **puntero** es una referencia a un objeto.
- Una **variable puntero** es una variable cuyo r-valor es una referencia a un objeto.

# TIPOS DE DATOS

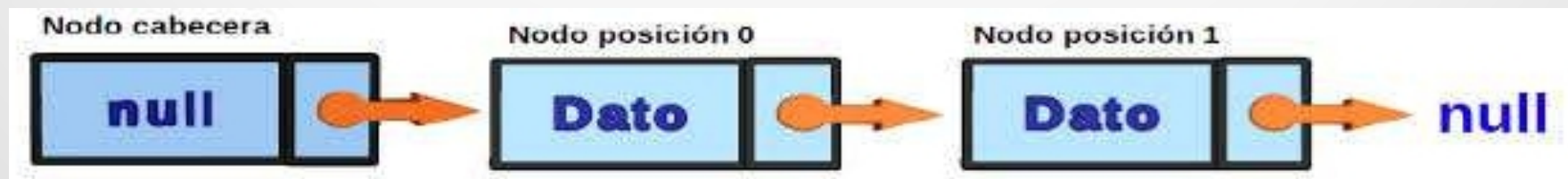
PUNTEROS Y MANEJO DE MEMORIA

# TIPOS DE DATOS – PUNTEROS

Estructuras de tamaño arbitrario, número de items no determinado: los punteros permiten conectar juntos muchos items sin tener un nombre explícito para todos ellos. (recursión)

relaciones múltiples entre los items: los punteros permiten que el dato sea puesto en varias estructuras sin necesidad de duplicarlo.

Acceso a bajo nivel: los punteros están cerca de la máquina

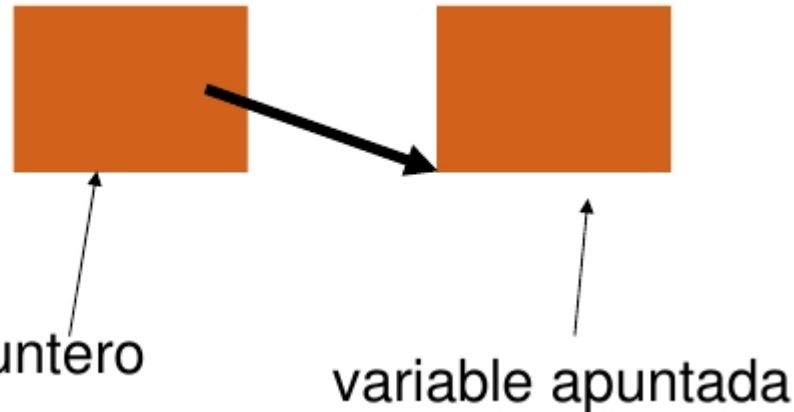


# TIPOS DE DATOS – PUNTEROS

Recursión

VALORES:

- direcciones de m
- valor nulo (no así variable puntero)



OPERACIONES (l-valor r-valor r-valor de la variable apuntada)

- Asignación de valor: generalmente asociado a la alocaión de la variable apuntada
- Referencias: a su valor (como dirección), operaciones entre punteros al valor de la variable apuntada: desreferenciación implícita

# TIPOS DE DATOS – PUNTEROS

Los punteros son un mecanismo muy potente para definir estructuras de datos recursivas.

Por acceder a bajo nivel, pueden obscurecer o hacer **inseguros** a los programas que los usan.

Se compara a los punteros con los *go to*:

- Los *go to* amplían el rango de sentencias que pueden ejecutar.
- Los punteros amplían el rango de las celdas de memoria que pueden ser referenciadas por una variable y también amplían el tipo de los valores que puede contener un objeto.

# TIPOS DE DATOS – PUNTEROS

## PUNTEROS

### INSEGURIDAD DE LOS PUNTEROS

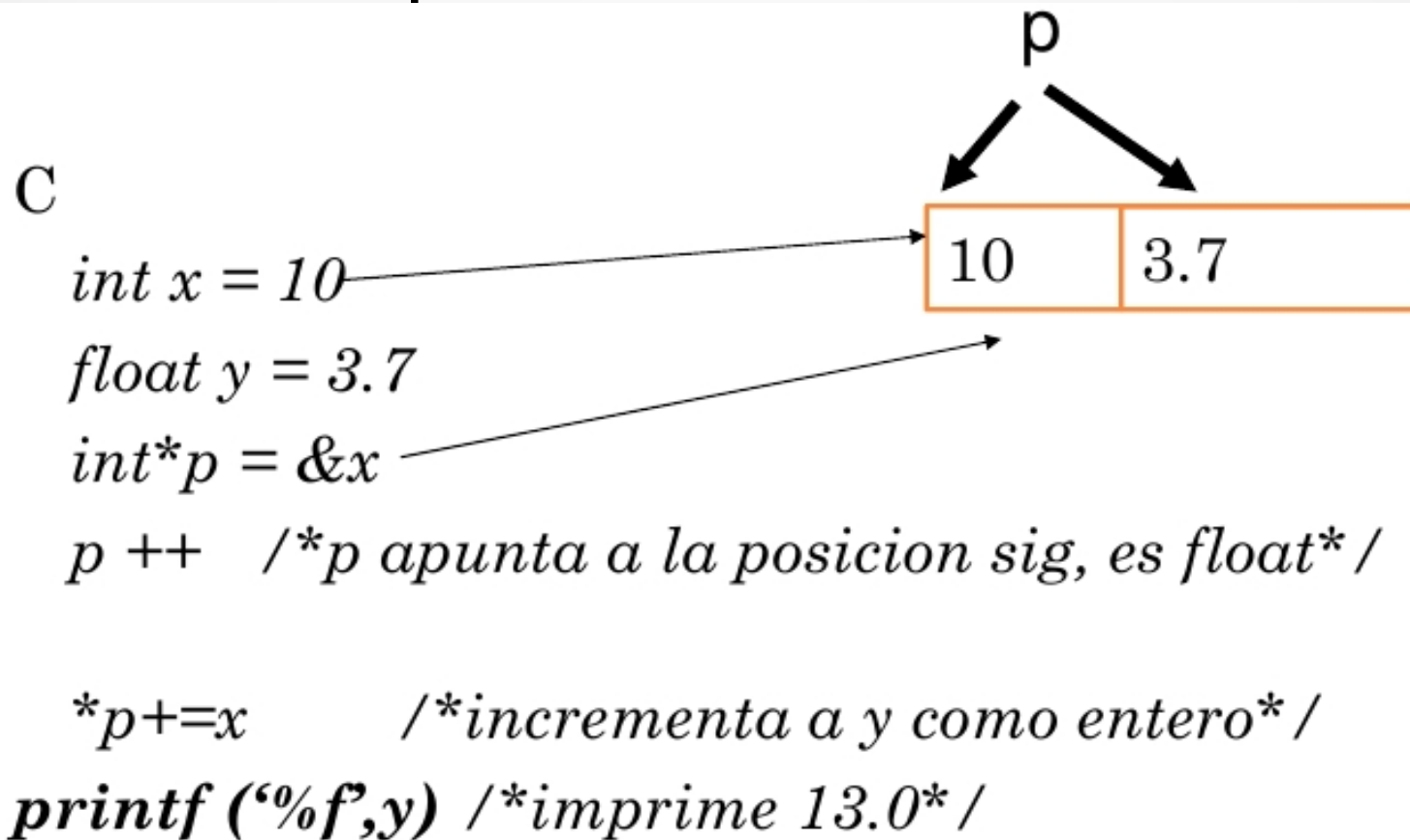
1. Violación de tipos
2. Referencias sueltas – referencias dangling
3. Punteros no inicializados
4. Punteros y uniones discriminadas
5. Alias
6. Liberación de memoria: objetos perdidos



# TIPOS DE DATOS – PUNTEROS

## INSEGURIDAD DE LOS PUNTEROS

### 1. Violación de tipos



# TIPOS DE DATOS – PUNTEROS

## INSEGURIDAD DE LOS PUNTEROS

### 2. Referencias sueltas – referencias dangling

Si este objeto no esta alocado se dice que el puntero es peligroso (dangling).

Una referencia suelta o dangling es un puntero que contiene una dirección de una variable dinámica que fue desalocada. Si luego se usa el puntero producirá error.

# TIPOS DE DATOS – PUNTEROS

## INSEGURIDAD DE LOS PUNTEROS

### 3. Punteros no inicializados

Peligro de acceso descontrolado a posiciones de memoria

Verificación dinámica de la inicialización

Solución: valor especial nulo:

- nil en Pascal
- void en C/C++
- null en ADA, Python

# TIPOS DE DATOS – PUNTEROS

## INSEGURIDAD DE LOS PUNTEROS

### 4. Punteros y uniones discriminadas

```
union ojo{  
  int int_var  
  int* int_ref}
```

En el caso de C, este es el mismo efecto que causa la aritmética de punteros. Para resolver este problema asociado con los punteros Java elimina la noción de puntero explícito completamente.

# TIPOS DE DATOS – PUNTEROS

## INSEGURIDAD DE LOS PUNTEROS

### 5.Alias

p1 y p2 son punteros  
p1 y x son alias  
p2 y x también lo son

```
int* p1
```

```
int* p2
```

```
int x
```

```
p1 = &x
```

```
p2 = &x
```

# TIPOS DE DATOS – PUNTEROS

## INSEGURIDAD DE LOS PUNTEROS

### 6. Liberación de memoria: objetos perdidos

Las variables puntero se alocan como cualquier otra variable en la pila de registros de activación

- Los objetos (apuntados) que se alocan a través de la primitiva `new` son alocados en la heap.
- La memoria disponible (heap) podría rápidamente agotarse a menos que de alguna forma se devuelva el almacenamiento alocado liberado

# TIPOS DE DATOS — MANEJO DE MEMORIA

Si los objetos en el heap dejan de ser accesibles  
esa memoria podría liberarse

garbage (objetos perdidos)

Un objeto se dice accesible si alguna variable en  
la pila lo apunta directa o indirectamente.

Un objeto es basura si no es accesible.

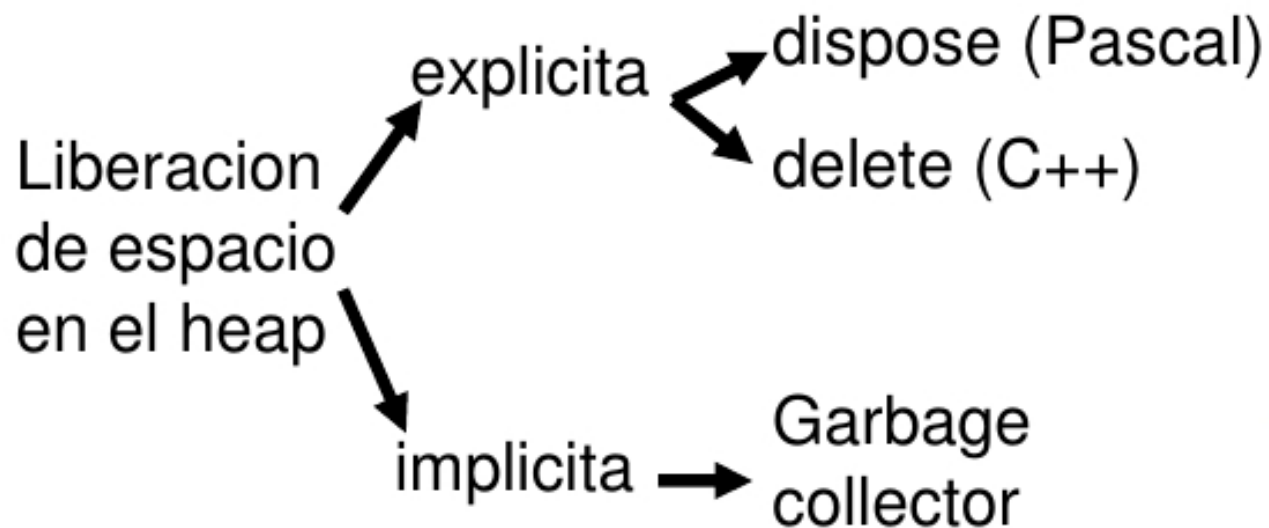
Mecanismos para desalocar memoria

# TIPOS DE DATOS – MANEJO DE MEMORIA

## Liberación de Memoria

reconocimiento de que  
porción de la memoria es  
basura

se requiere o no  
intervención del usuario



Responsabilidad  
del programador

Independiente de  
la aplicación



# TIPOS DE DATOS — MANEJO DE MEMORIA

El reconocimiento de la basura recae en el programador, quien notifica al sistema cuando un objeto ya no se usa.

No garantiza que no haya otro puntero que apunte a esta dirección definida como basura, este puntero se transforma en dangling. (puntero suelto).

Este error es difícil de chequear y la mayoría de los lenguajes no lo implementan por que es costoso.

# TIPOS DE DATOS – MANEJO DE MEMORIA

- Liberación de Memoria

## EXPLICITA: PASCAL

- Proceso llamado *dispose* libera memoria en el heap
- Puede generar referencias sueltas
- Para evitarlo se necesitaría una verificación dinámica para garantizar el uso correcto del *dispose*

*new (p)*

*q := p*

*dispose (p)*

*...q ...*

libera el espacio del nodo  
y pone p en nil

q referencia  
suelta

# TIPOS DE DATOS — MANEJO DE MEMORIA

- Liberación de Memoria

## Implícita

El sistema, durante la ejecución tomará la decisión de descubrir la basura por medio de un algoritmo de recolección de basura: garbage collector.

Muy importante para los lenguajes de programación que hacen un uso intensivo de variables dinámicas.(LISP, Python).

# TIPOS DE DATOS – MANEJO DE MEMORIA

- Liberación de Memoria

Implícita

Se ejecuta durante el procesamiento de las aplicaciones

Sistema interactivo o de tiempo real: no bajar en el rendimiento y evitar los riesgos.

Debe ser muy eficiente

Debe ejecutar en paralelo

# TIPOS DE DATOS — MANEJO DE MEMORIA

- Liberación de Memoria

## Implícita

- En Algol-68 y Simula 67 el garbage collector reclama automáticamente la memoria no utilizada.
- Eiffel y Java que uniformemente tratan a todos los objetos como referenciados por punteros proporcionan un recolector automático.
- ADA chequea dinámicamente que el tiempo de vida de los objetos apuntados sea menor igual que el del puntero.
- PHP chequea una cuenta de referencias al objeto.

# TIPOS DE DATOS – MANEJO DE MEMORIA

- Liberación de Memoria - Garbage Collector – Comparación de Implementaciones

	Java	PHP
Mecanismo	Generacional	Cuenta de Referencias
Reconocimiento	Divide la Heap en 3 generaciones (Joven / Vieja y Permanente)	Contenedor z-val con 2 campos: is_ref y refcount
Algoritmo	Se van envejeciendo y compactando los objetos referenciados. Se eliminan objetos sin referencia y se organizan los bloques de generaciones.	Se cuenta la cantidad de referencias de cada objeto y al llegar a 0, si no es referenciado se elimina
Configuraciones	Serial GC – Parallel GC - CMS GC – Garbage First	Fallo con fuga de memoria solucionado a partir de 5.3.0

# TIPOS DE DATOS

## TIPO DE DATO ABSTRACTO

# TIPOS DE DATOS - TADs

## ABSTRACCIÓN

La abstracción es el mecanismo que tenemos las personas para manejar la complejidad

Abstraer es representar algo descubriendo sus características esenciales y suprimiendo las que no lo son.

El principio básico de la abstracción es la **información oculta**.



# TIPOS DE DATOS - TADs

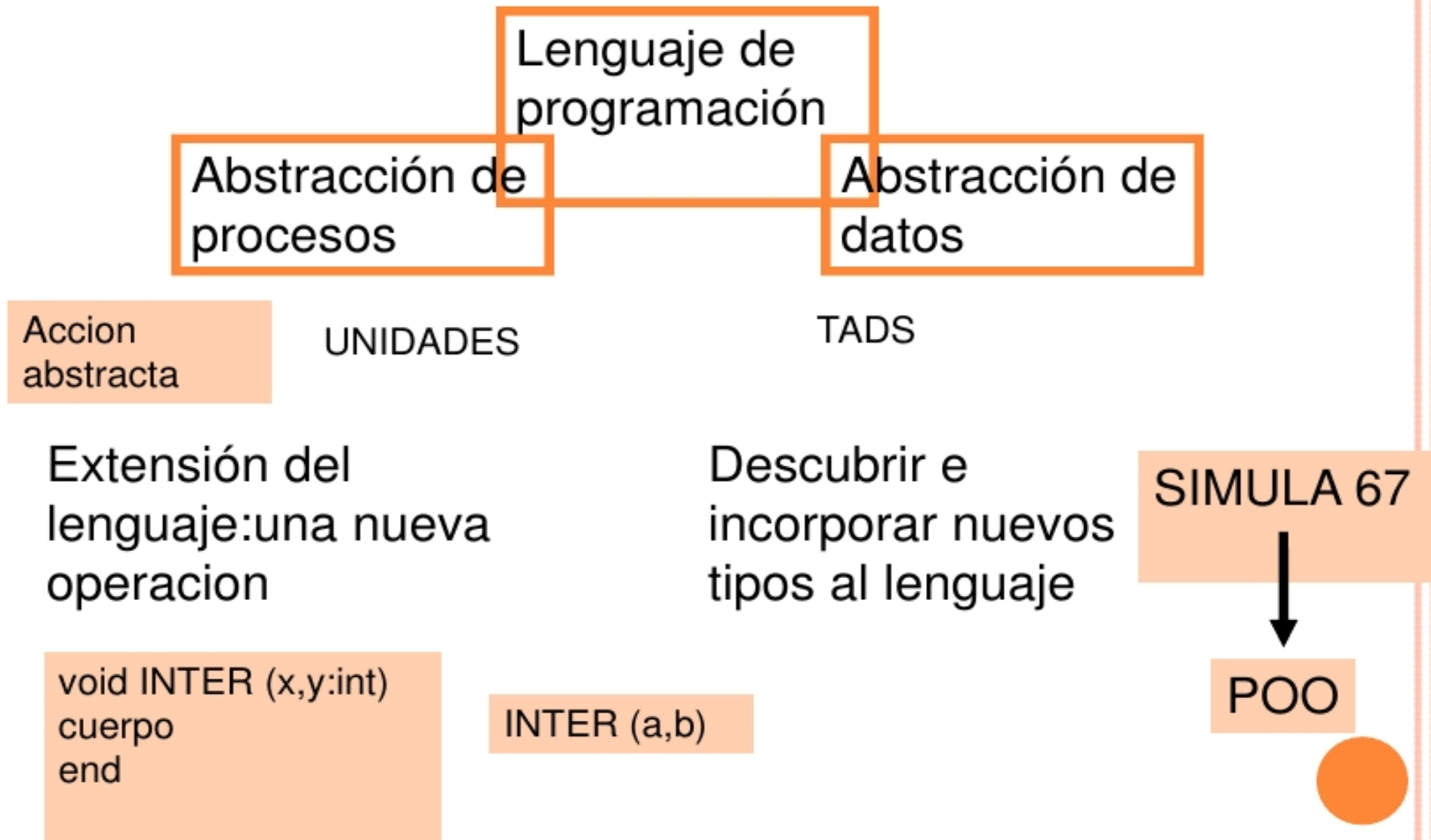
TAD

**TAD = Representación (datos) + Operaciones  
(funciones y procedimientos)**

Los tipos de datos son abstracciones y el proceso de construir nuevos tipos se llama **abstracción de datos**.

Los nuevos tipos de datos definidos por el usuario se llaman **tipos abstractos de datos**.

# TIPOS DE DATOS - TADs



# TIPOS DE DATOS - TADs

## TAD

Tipo abstracto de dato (TAD) es el que satisface:

- Encapsulamiento: la representación del tipo y las operaciones permitidas para los objetos del tipo se describen en una única unidad sintáctica.

Refleja las abstracciones descubiertas en el diseño

- Ocultamiento de la información: la representación de los objetos y la implementación del tipo permanecen ocultos.

Refleja los niveles de abstracción. Modificabilidad

# TIPOS DE DATOS - TADs

## TAD

Las unidades de programación de lenguajes que pueden implementar un TAD reciben distintos nombres:

- Simula-67 proporciona una estructura sintáctica que permite que las operaciones y la representación puedan especificarse en una única unidad sintáctica (class). Pero no satisface el ocultamiento de la información.
- Modula-2 módulo
- Ada paquete
- C++ clase
- Java clase

# TIPOS DE DATOS - TADs

## ESPECIFICACIÓN DE UN TAD

La especificación formal proporciona un conjunto de axiomas que describen el comportamiento de todas las operaciones.

Ha de incluir una parte de sintaxis y una parte de semántica. Por ejemplo:

TAD nombre del tipo (valores que toma los datos del tipo)

Sintaxis

Operación(Tipo argumento, ...)  $\rightarrow$  Tipo resultado

....

Semántica

Operación(valores particulares argumentos)  $\Rightarrow$  expresión resultado

Hay operaciones definidas por sí mismas que se consideran constructores del TAD. Normalmente, se elige como constructor la operación que inicializa.

# TIPOS DE DATOS - TADs

## EJEMPLO TAD: CONJUNTO

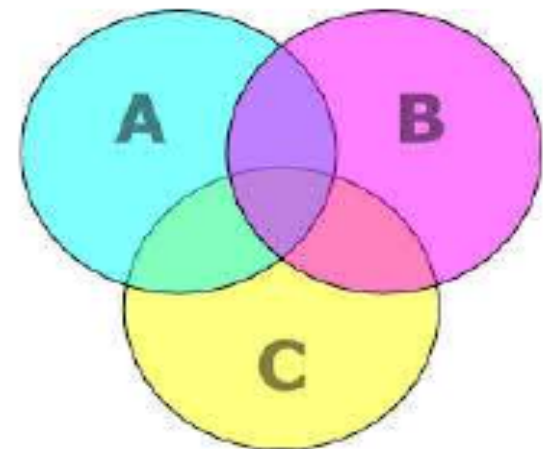
- *TAD Conjunto (colección de elementos sin duplicidades, pueden estar en cualquier orden, se usa para representar los conjuntos matemáticos con sus operaciones).*

### Sintaxis

- $*\text{Conjuntovacio} \rightarrow \text{Conjunto}$
- $*\text{Añadir}(\text{Conjunto}, \text{Elemento}) \rightarrow \text{Conjunto}$
- $\text{Retirar}(\text{Conjunto}, \text{Elemento}) \rightarrow \text{Conjunto}$
- $\text{Pertenece}(\text{Conjunto}, \text{Elemento}) \rightarrow \text{boolean}$
- $\text{Esvacio}(\text{Conjunto}) \rightarrow \text{boolean}$
- $\text{Cardinal}(\text{Conjunto}) \rightarrow \text{entero}$
- $\text{Union}(\text{Conjunto}, \text{Conjunto}) \rightarrow \text{Conjunto}$

**Semántica**  $\forall e1, e2 \in \text{Elemento}$  y  $\forall C, D \in \text{Conjunto}$

- $\text{Añadir}(\text{Añadir}(C, e1), e1) \Rightarrow \text{Añadir}(C, e1)$
- $\text{Añadir}(\text{Añadir}(C, e1), e2) \Rightarrow \text{Añadir}(\text{Añadir}(C, e2), e1)$
- $\text{Retirar}(\text{Conjuntovacio}, e1) \Rightarrow \text{Conjuntovacio}$
- $\text{Retirar}(\text{Añadir}(C, e1), e2) \Rightarrow$  si  $e1 = e2$  entonces  $\text{Retirar}(C, e2)$   
sino  $\text{Añadir}(\text{Retirar}(C, e2), e1)$



# TIPOS DE DATOS - TADs

## EJEMPLO TAD: PILA EN ADA

- PILA de enteros (100)

ESPECIFICACION

- package PILA IS
- type PILA limited private
- MAX: constant := 100
- function EMPTY (P:in PILA) return boolean
- procedure PUSH (P:inout PILA,ELE:in INTEGER)
- procedure POP (P: inout PILA)
- procedure TOP (P:inPILA) return INTEGER

ENCAPSULA

- private
- type PILA is
- vecpila : array (1..MAX) of INTEGER
- tope: INTEGER range 0..MAX:=0
- end PILA

OCULTA



# TIPOS DE DATOS - TADs

## IMPLEMENTACION

```
○ package body PILA is
○   function EMPTY (P:in PILA) return boolean
○       .....
○   end
○   procedure PUSH (P:inout PILA,ELE:in INTEGER)
○       .....
○   end
○   procedure POP (P: inout PILA)
○       .....
○   end
○   procedure TOP (P:inPILA) return INTEGER
○       .....
○   end
○ end PILA
```

OCULTA





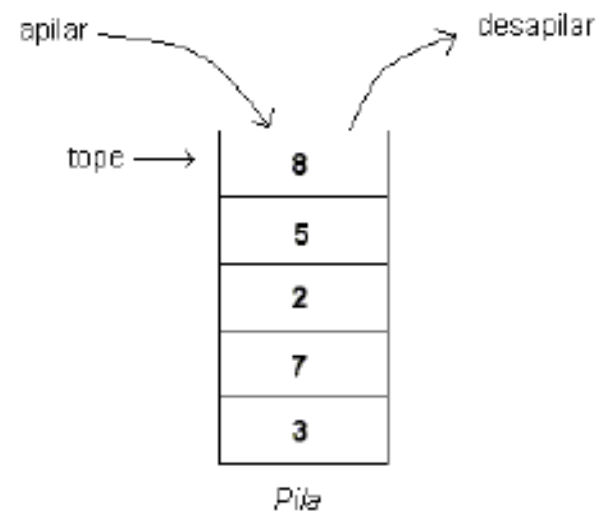
# TIPOS DE DATOS - TADs

## INSTANCIACION DE UNA PILA

```
with PILA
procedure USAR
  pil:PILA
  y: INTEGER
  .....
  pil.PUSH (pil,y)
End USAR
```

INSTANCIA  
ALOCA Y EJECUTA  
EL CODIGO DE  
INICIALIZACION

APILA



# TIPOS DE DATOS - TADs

## TAD: CLASES

En términos prácticos, una clase es un tipo definido por el usuario

Una clase contiene la especificación de los datos que describen un objeto junto con la descripción de las acciones que un objeto conoce.

### Atributos + Métodos

Agrega un segundo nivel de abstracción que consiste en agrupar las clases en jerarquías de clases. De forma que la clase hereda todas las propiedades de la superclase.

# TIPOS DE DATOS - TADs

## EJEMPLO: CLASE EN JAVA

```
class Punto
{ private int x; // coordenada x
  private int y; // coordenada y
  public Punto(int x_, int y_) // constructor
  { x = x_; y = y_; }
  public Punto() // constructor sin argumentos
  { x = y = 0; }
  public int leerX() // devuelve el valor de x
  { return x; }
  public int leerY() // devuelve el valor de y
  { return y; }
  void fijarX(int valorX) // establece el valor de x
  { x = valorX; }
  void fijarY(int valorY) // establece el valor de y
  { y = valorY; }
}
```

Visibilidad de los miembros de la clase

```
Punto p;
p = new Punto();
p.fijarX(100);
System.out.println("Coordenada x es " + p.leerX());
```

# TIPOS DE DATOS

## SISTEMA DE TIPOS

# TIPOS DE DATOS – SISTEMA DE TIPOS

## SISTEMA DE TIPOS

Conjunto de reglas usadas por un lenguaje para estructurar y organizar sus tipos.

El objetivo de un sistema de tipos es escribir programas seguros.

Conocer el sistema de tipos de un lenguaje nos permite conocer de una mejor forma los aspectos semánticos del lenguaje.

# TIPOS DE DATOS – SISTEMA DE TIPOS

## SISTEMA DE TIPOS

- Provee mecanismos de expresión:
  - Expresar tipos intrínsecos o definir tipos nuevos.
  - Asociar los tipos definidos con construcciones del lenguaje.
- Define reglas de resolución:
  - Equivalencia de tipos – ¿dos valores tienen el mismo tipo?.
  - Compatibilidad de tipos – ¿puedo usar el tipo en este contexto?
  - Inferencia de tipos – ¿cuál tipo se deduce del contexto?
- Mientras más flexible el lenguaje, más complejo el sistema

SEGURIDAD VS. FLEXIBILIDAD

# TIPOS DE DATOS – SISTEMA DE TIPOS

## TIPADO FUERTE – TIPADO DÉBIL

Se dice que el sistema de tipos es fuerte cuando especifica restricciones sobre como las operaciones que involucran valores de diferentes tipos pueden operarse. Lo contrario establece un sistema débil de tipos.

```
a = 2
```

```
b= "2"
```

```
Concatenar (a,b) //retorna "22"
```

```
Sumar (a,b) //retorna 4
```

```
a = 2
```

```
b= "2"
```

```
Concatenar (a,b) //error de tipos
```

```
Sumar (a,b) //error de tipos
```

```
Concatenar (str(a),b) //retorna "22"
```

```
Sumar (a,int(b)) //retorna 4
```

# TIPOS DE DATOS – SISTEMA DE TIPOS

## SISTEMA DE TIPOS - ESPECIFICACIÓN

- Tipo y tiempo de chequeo
- Reglas de equivalencia y conversión
- Reglas de inferencia de tipo
- Nivel de polimorfismo del lenguaje



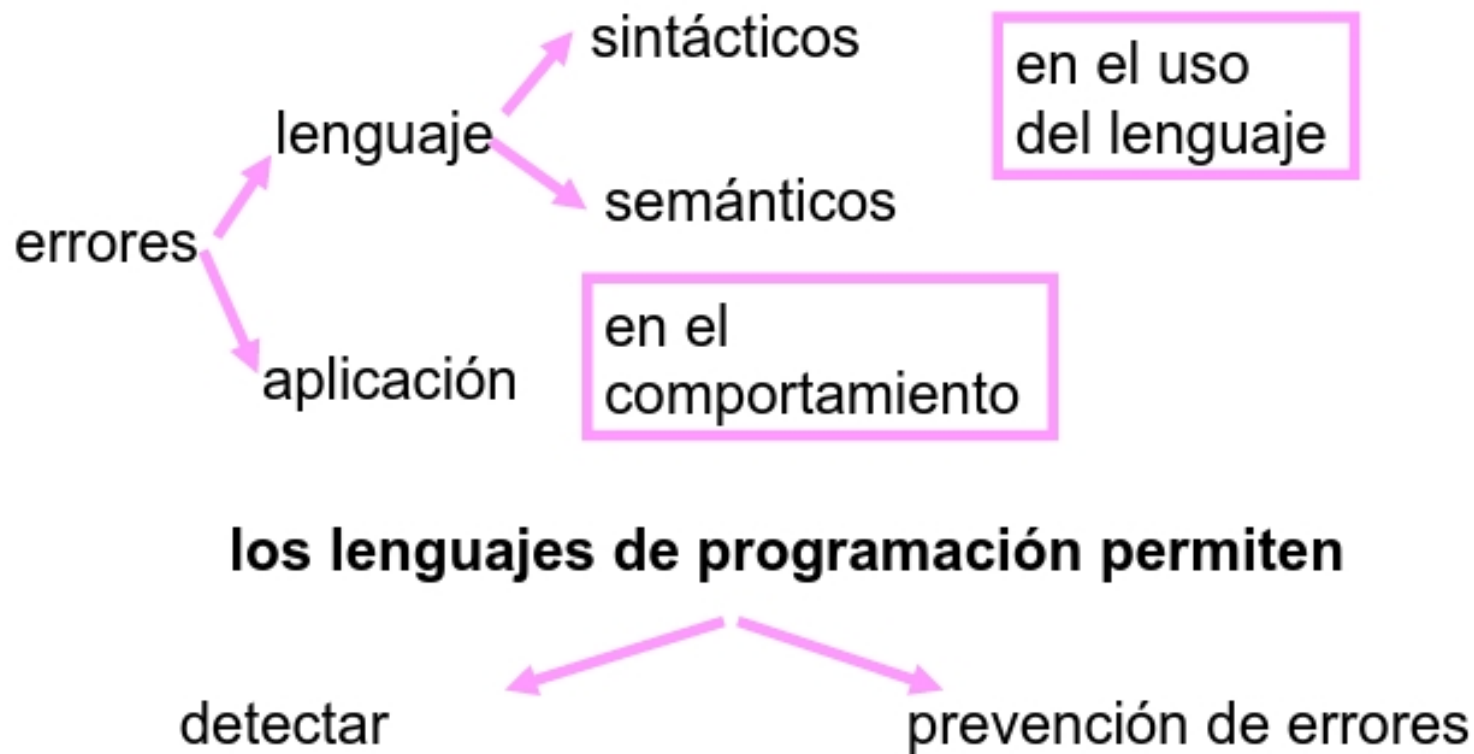
# TIPOS DE DATOS – SISTEMA DE TIPOS

Tipo y Tiempo de Chequeo

Tipos de ligadura

- Tipado estático: ligaduras en compilación. Para esto puede exigir:
  - Se puedan utilizar tipos de datos predefinidos
  - Todas las variables se declaran con un tipo asociado
  - Todas las operaciones se especifican indicando los tipos de los operandos requeridos y el tipo del resultado.
- Tipado dinámico: ligaduras en ejecución, provoca mas comprobaciones en tiempo de ejecución (no es seguro???)
- Tipado seguro: no es estático, ni inseguro

# TIPOS DE DATOS – SISTEMA DE TIPOS



# TIPOS DE DATOS – SISTEMA DE TIPOS

## Lenguaje Fuertemente Tipado

- Si el lenguaje es fuertemente tipado el compilador puede garantizar la ausencia de errores de tipo en los programas (Ghezzi).
- Un lenguaje se dice fuertemente tipado (type safety) si el sistema de tipos impone restricciones que aseguran que no se producirán errores de tipo en ejecución.
- Un lenguaje se dice fuertemente tipado (type safety) si todos los errores de tipo se detectan.

# TIPOS DE DATOS – SISTEMA DE TIPOS

## Lenguaje Fuertemente Tipado

En esta concepción, la intención es evitar los errores de aplicación y son tolerados los errores del lenguaje, detectados tan pronto como sea posible.

Python es Fuertemente Tipado y tiene tipado dinámico.  
C es débilmente tipado y tiene tipado estático.  
GOBSTONE Fuertemente Tipado y tipado estático.

# TIPOS DE DATOS – SISTEMA DE TIPOS

## Reglas de Equivalencia y Conversión

- Tipo compatible: reglas semánticas que determinan si el tipo de un objeto es válido en un contexto particular
- Un lenguaje debe definir en que contexto el tipo  $Q$  es compatible con el tipo  $T$ .
- Si el sistema de tipos define la compatibilidad.

# TIPOS DE DATOS – SISTEMA DE TIPOS

## Reglas de Equivalencia y Conversión

*type t = array [1..100] of integer*

*var x,y : array [1..100] of integer*

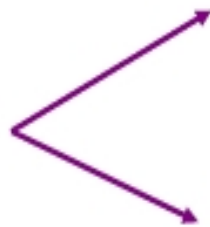
*z: array [1..100] of integer*

*w:t*

*v:t*

w, x, y, z, v son del mismo tipo??

**Compatibilidad**



*Nombre*

*Estructura*

# TIPOS DE DATOS – SISTEMA DE TIPOS

## Reglas de Equivalencia y Conversión

- Equivalencia por nombre: dos variables son del mismo tipo si y solo si están declaradas juntas o si están declaradas con el mismo nombre de tipo.
- Equivalencia por estructura: dos variables son del mismo tipo si los componentes de su tipo son iguales.

# TIPOS DE DATOS – SISTEMA DE TIPOS

## Reglas de Equivalencia y Conversión

- Un tipo es compatible con otro si
  - es equivalente
  - se puede convertir
- Coerción: significa convertir un valor de un tipo a otro. Reglas del lenguaje de acuerdo al tipo de los operandos y a la jerarquía



# TIPOS DE DATOS – SISTEMA DE TIPOS

## Reglas de Equivalencia y Conversión

- Widening (ensanchar): cada valor del dominio tiene su correspondiente valor en el rango. (Entero a Real). Pascal solo widening de entero a real.
- Narrowing (estrechar): cada valor del dominio puede no tener su correspondiente valor en el rango. En este caso algunos lenguajes producen un mensaje avisando la pérdida de información. (Real a Entero). En C Depende del contexto y utiliza un sistema de coersión simple.
- Cláusula de casting : conversiones explicitas, se fuerza a que se convierta.

# TIPOS DE DATOS – SISTEMA DE TIPOS

## Reglas de Inferencia y nivel de polimorfismo

La inferencia de tipos permite que el tipo de una entidad declarada se “infiera” en lugar de ser declarado. La inferencia puede realizarse de acuerdo al tipo de:

- Un operador predefinido  
fun f1(n,m)=(n mod m=0)

- Un operando  
fun f2(n) = (n\*2)

- Un argumento  
fun f3(n:int) = n\*n

- El tipo del resultado  
fun f4(n):int = (n\*n)

# TIPOS DE DATOS – SISTEMA DE TIPOS

## Reglas de Inferencia y nivel de polimorfismo

Veamos los siguientes ejemplos:

```
write(e,f(x)+1)
```

```
fun disjuntos(s1,s2:Conjunto):boolean;
```

- La función disjuntos deberá implementarse para cada tipo particular de conjunto?
- Las funciones read y write son “polimórficas”, pero no de forma pura (ya que el compilador infiere el tipo)

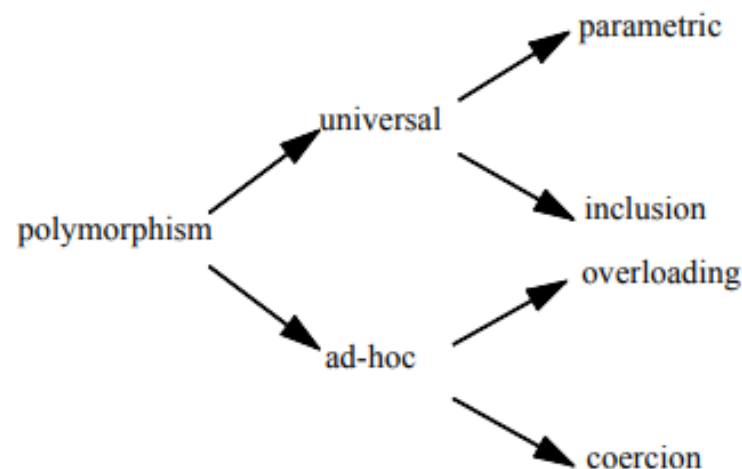
# TIPOS DE DATOS – SISTEMA DE TIPOS

- Un lenguaje se dice mono-mórfico si cada entidad se liga a un único tipo (estáticos). En un lenguaje de programación mono-mórfico la función disjuntos deberá implementarse para cada tipo particular de conjunto.
- Un lenguaje se dice **polimórfico** si las entidades pueden estar ligadas a más de un tipo
- Las variables polimórficas pueden tomar valores de diferentes tipos
- Las operaciones polimórficas son funciones que aceptan operandos de varios tipos
- Los tipos polimórficos tienen operaciones polimórficas

# TIPOS DE DATOS – SISTEMA DE TIPOS

## Reglas de Inferencia y nivel de polimorfismo

Todos los lenguajes prácticos tienen cierto grado de polimorfismo. En consecuencia, las preguntas importantes a responder son: ¿Qué diferentes tipos (o grados) de polimorfismo se identifican? ¿Hasta dónde podemos llegar?



# TIPOS DE DATOS – SISTEMA DE TIPOS

## Reglas de Inferencia y nivel de polimorfismo

El **polimorfismo ad-hoc** permite que una función se aplique a distintos tipos con un comportamiento sustancialmente diferente en cada caso

El término **sobrecarga** se utiliza para referirse a conjuntos de abstracciones diferentes que están ligadas al mismo símbolo o identificador.

La **coerción** permite que un operador que espera un operando de un determinado tipo T puede aplicarse de manera segura sobre un operando de un tipo diferente al esperado

# TIPOS DE DATOS – SISTEMA DE TIPOS

Reglas de Inferencia y nivel de polimorfismo

El **polimorfismo universal** permite que una única operación se aplique uniformemente sobre un conjunto de tipos relacionados

Si la uniformidad de la estructura de tipos está dada a través de parámetros, hablamos de **polimorfismo paramétrico**

Un tipo parametrizado es un tipo que tiene otros tipos como parámetros

$\text{lista}(T) = T^*$

El **polimorfismo por inclusión** es otra forma de polimorfismo universal que permite modelar subtipos y herencia.

- Si un tipo se define como un conjunto de valores y un conjunto de operaciones. Un subtipo  $T'$  de un tipo  $T$  puede definirse como un subconjunto de los valores de  $T$  y el mismo conjunto de operaciones.

```
subtype TDíaDelMes is Integer range 1..31;  
subtype TDíaFebrero is TDíaDelMes range 1..29;  
subtype TLaborable is TDíaDeSemana range Lunes..Viernes;
```

- El mecanismo de herencia permite definir una nueva clase derivada a partir de una clase base ya existente. Podría agregar atributos y comportamiento.