ANÁLISIS COMPARATIVO DE LENGUAJES

Guía de estudio correspondiente a la Teoría sobre: Tipos de datos, Chequeo de Tipos, Declaraciones, Operaciones, Tipos de Datos Elementales

Depto. de Informática

Universidad Nacional de San Luis - Año 2022

TIPOS DE DATOS

Notas de Clase

Capítulos V y VI Programming Languages – Design and Implementation – Terrence Pratt

OBJETIVOS DE ESTA TEORÍA

Estudiremos:

- Objetos de Datos y Tipos de Datos.
- Especificación e Implementacion de Tipos de Datos. Atributos – Operaciones- Declaraciones
- Chequeo de Tipos.
- Tipos de Datos Elementales (TDE).

INTRODUCCIÓN

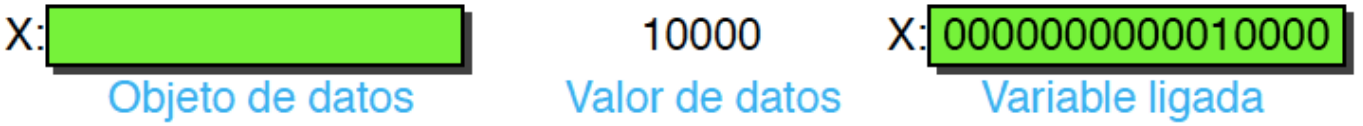
Cualquier programa, independientemente del lenguaje, se puede considerar como la especificación de un conjunto de operaciones, que se van a aplicar a ciertos datos, en un orden determinado.
Es por ello que comenzaremos a estudiar los datos, tipos y operaciones que se incorporan comúnmente en los lenguajes de programación.

OBJETOS DE DATOS (OD)

Definición: Ubicación en memoria utilizada en tiempo de ejecución para almacenar valores de datos.

Objeto de datos ≡ recipiente para valores de datos.

Ejemplo: `x = 16;`



Objeto de datos — Valor de datos — Variable ligada

OBJETOS DE DATOS

- Los objetos de datos que existen durante la ejecución de programas pueden ser:

OBJETOS DE DATOS

Un objeto de datos puede ser:
Elemental: contiene un valor de datos que siempre se manipula como una unidad.
Estructurado (o estructura de datos): si está compuesto de otros objetos de datos que pueden ser elementales o estructurados.
Cada objeto de datos tiene asociado lo que se denomina **tiempo de vida**.
Tiempo de vida de un objeto de datos: tiempo durante el cual un objeto de datos *existe* (tiene asignada una ubicación de memoria y puede ser usado para almacenar valores de datos).

OBJETOS DE DATOS ATRIBUTOS Y LIGADURAS

Atributos: propiedades asociadas a un OD, que se mantienen invariantes durante su tiempo de vida.

El **tipo** de un OD es un atributo, que se determina normalmente en tiempo de traducción del programa.

Ligaduras: asociaciones en las que participa un OD, pueden cambiar dinámicamente durante la ejecución del programa.

LIGADURAS DE UN OD

Un OD puede participar en **varias ligaduras** (asociaciones) durante su tiempo de vida, las ligaduras más importantes son:

- Ubicación
- Valor
- Nombre

VARIABLES Y CONSTANTES

Variable: OD que el programador define y nombra explícitamente en un programa.
Variable simple: OD elemental con un nombre. La ligadura de una variable a un valor puede ser modificada mediante operaciones de asignación.
Descriptor: Un descriptor representa la colección de atributos de una variable.
Constante: OD con un nombre ligado a un valor en forma permanente durante todo su tiempo de vida.

CONSTANTES

Constante literal (21, `"hola"`, 56,79)

Constante definida por el programador (`const int MAX=30`)

Constantes en lenguaje C:

```
const int JUGADORES = 5;
  # define JUGADORES 5
```

¿Cuál es la diferencia entre una declaración y la otra?

En la primera declaración se indica que `JUGADORES` es una constante de tipo `int` mientras que en la segunda se indica que donde aparezca en el código la palabra `JUGADORES` deberá ser reemplazada por 5 directamente.

TIPOS DE DATOS

Definición: clase de OD's con su propio conjunto de operaciones para crearlos y manipularlos.

Todo lenguaje tiene un conjunto de tipos de datos primitivos (los provistos directamente por el lenguaje) y en general todos los lenguajes modernos permiten la creación de nuevos tipos (`type` en Pascal, `typedef` en C, `class` en Java, etc.).

El estudio de los tipos de datos involucra los siguientes aspectos: **especificación** e **implementación.**

ESPECIFICACIÓN E IMPLEMENTACIÓN DE TIPOS DE DATOS

Especificación:

- Sus *atributos* que distinguen a los objetos de dicho tipo.
- Los *valores* posibles que pueden tener los objetos de dichos tipos.
- Las *operaciones* válidas para manipular los objetos de dicho tipo.

Implementación:

- La *representación en memoria* que se usa para la representación de los OD.
- Los *algoritmos* que implementan las operaciones del tipo de datos.

OPERACIONES

LAS OPERACIONES MERECEN ESPECIAL ATENCIÓN....

Definición: es una función matemática que puede ser definida especificando el tipo de los argumentos de entrada (su dominio) y el tipo de los resultados (su rango).

Signatura (o prototipo) de una operación: especifica el número, orden y tipos de datos de los argumentos y los resultados:

$$\text{nombre operación: tipo arg.} \times \ldots \times \text{tipo arg.} \rightarrow \text{tipo resultado}$$

Ejemplo: $+$: entero x entero $\rightarrow$ entero (suma de enteros)

PROTOTIPOS EN LENGUAJE C

```
/* Prototipo */
int addition(int, int);
/* Función principal */
int main()
{
    int i, j;

    i = 10;
    j = 20;
    /* Invocación de la función */
    i += addition(i, j);
}
/* Definición de la función */
int addition(int a, int b)
{
    return (a + b);
}
```

PROBLEMAS AL ESPECIFICAR OPERACIONES

Sin embargo no siempre es sencillo determinar una especificación precisa, como una función matemática, para toda operación.

Problemas:

- Las operaciones pueden estar **indefinidas** para ciertas entradas.
- **Argumentos implícitos**.
- **Efectos colaterales** (resultados implícitos).
- **Auto modificación** (sensitividad a la historia)

EJEMPLOS

```
int a, b, c, sum;
 int suma (int x, int y) {
   return x + y + a;
}
void main () {
   scanf (“%i”,&a,“%i”,&b,“%i”,&c);
   sum = suma (b,c);
}
```

```
int a, b, c, sum;
 int suma (int x, int y) {
   a = x + y;
   return x + y + a;
}
void main () {
   scanf (“%i”,&a,“%i”,&b,“%i”,&c);
   sum = suma (b,c);
}
```

```
int contar (void)
{
static long cuenta = 0;
int i = 10;
cuenta++;
printf("Llamada \%ld veces\n", cuenta);
return (cuenta/i);
}
```

IMPLEMENTACIÓN DE TIPOS DE DATOS ELEMENTALES

La implementación de un tipo elemental de datos se compone de una *representación de almacenamiento* para objetos de datos y valores de ese tipo, y un *conjunto de algoritmos* que definen las operaciones.
Representación en memoria: si un tipo de datos es soportado por el hardware subyacente, se utiliza la misma representación de memoria. En otro caso, deberán ser simuladas por software.

DECLARACIONES

Cuando un programador escribe un programa, determina el nombre y tipo de cada uno de los OD necesarios, esto lo hace a través de las **declararaciones**.

Una **declaración** es una sentencia del programa que sirve para comunicarle al traductor del lenguaje, información sobre el nombre y tipo de los OD que se necesitan durante la ejecución del programa (y también de sus tiempos de vida).

Las declaraciones pueden ser:

- Explícitas
- Implícitas

Ej: Fortran, Python y Perl proveen declaraciones implícitas. Fortran también posee la explícita.

PROPÓSITOS DE LAS DECLARACIONES

- Seleccionar una representación de memoria adecuada.
- Administración de memoria eficiente.
- Uso de operaciones polimórficas.
- Permiten chequeo de tipos estático

CHEQUEO DE TIPOS

El chequeo de tipos consiste en controlar que cada operación ejecutada por un programa reciba un número correcto de argumentos y del tipo correcto.

Momentos en que se puede realizar:

- En tiempo de ejecución: **chequeo de tipos dinámico.**
- En tiempo de traducción: **chequeo de tipos estático.**

CHEQUEO DE TIPOS DINÁMICO

- ✓ Normalmente se realiza inmediatamente antes de ejecutar una operación.
- ✓ Suele ser implementado agregando un campo con el tipo que actualmente tiene el objeto de datos.
- ✓ Los lenguajes sin declaraciones (con variables typeless) son diseñados para este tipo de chequeo. Ejemplo Lisp, Prolog, Smalltalk, JavaScript, PHP.
- ✓ Su principal ventaja es la flexibilidad, no requiere declaraciones y el tipo de un OD asociado con un nombre variable puede cambiar en ejecución.

CHEQUEO DE TIPOS DINÁMICO

Desventajas:

- Dificulta la depuración de programas.
- Requiere memoria extra.
- La velocidad de ejecución se reduce considerablemente.

CHEQUEO DE TIPOS ESTÁTICO

- ✓ Se realiza durante la traducción del programa.
- ✓ Se requiere la siguiente información:
  - ❖ Por cada operación, el número, orden y tipo de datos de los argumentos y resultados.
  - ❖ Por cada variables, el tipo de objeto de dato nombrado.
  - ❖ El tipo de cada objeto de dato constante.

CHEQUEO DE TIPOS ESTÁTICO

Ventajas:

- Todos los pasos posibles de ejecución son chequeados.
- No requiere información extra.
- No requiere controles en ejecución.

Desventajas:

- Es menos flexible.
- Algunas construcciones (en algunos lenguajes) no pueden ser chequeadas estáticamente.

LENGUAJES FUERTEMENTE TIPADOS

Un lenguaje es fuertemente tipado si los errores de tipos son siempre detectados.

C no es fuertemente tipado: si se declaran dos variables tipo short y se suman o multiplican, el resultado puede salirse del rango y causar un error.

Java es fuertemente tipado.

CONVERSIÓN DE TIPOS

La conversión de tipos es una operación con la signatura:

$$conversión\_op : tipo_1 \rightarrow tipo_2$$

Las conversiones de tipo pueden ser:

- En forma *explícita*: mediante construcciones y funciones invocadas explícitamente.
- En forma *implícita*: se realizan automáticamente (coerciones).

EJEMPLOS DE CONVERSIONES EXPLÍCITAS

En lenguaje C es posible realizar las siguientes conversiones:

```
int numeroentero;
float numeroflotante = 9.87;

numeroentero = (int) numeroflotante;
```

```
int numeroentero = 10;
float numeroflotante;

numeroflotante = (float) numeroentero;
```

```
int numeroentero;
char letra = 'A';

numeroentero = (int) letra;
```

EJEMPLOS DE CONVERSIONES IMPLÍCITAS (COERCIONES)

En lenguaje C es posible realizar las siguientes coerciones:

```
unsigned long int l;
float f;
double d;


f = l; /* l se convierte a float */
d = f; /* f se convierte a double */
f = d; /* d se reduce peligrosamente */
```

TIPOS DE DATOS ELEMENTALES

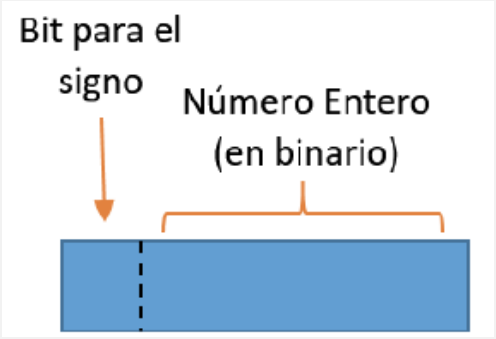
Nos concentraremos a partir de ahora en analizar la especificación e implementación de los siguientes tipos de datos elementales:

- Enteros
- Subrangos
- Reales
- Enumerados
- Booleanos
- Carácter
- String
- Punteros

ENTEROS

- Atributos: normalmente sólo el tipo.
- Valores: ordenados y dentro de ciertos límites.
- Operaciones: aritméticas, relacionales, asignación, operaciones de bits.
- Implementación: la representación y operaciones del hardware. La representación en memoria puede ser:

Sin descriptor: Se usa cuando el lenguaje provee chequeo de tipo estático y declaraciones de tipo.

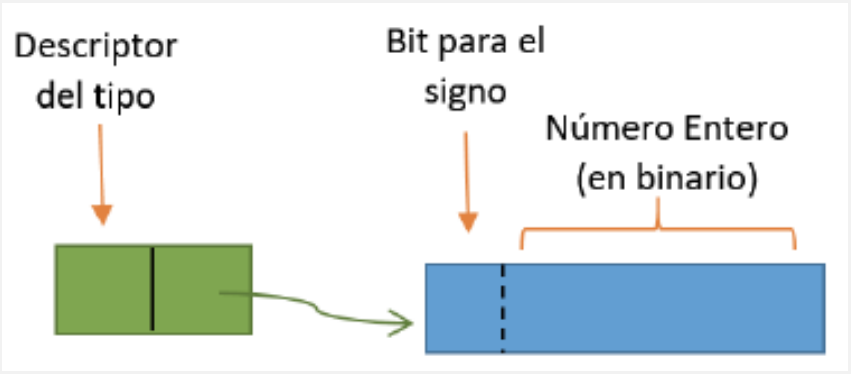


ENTEROS

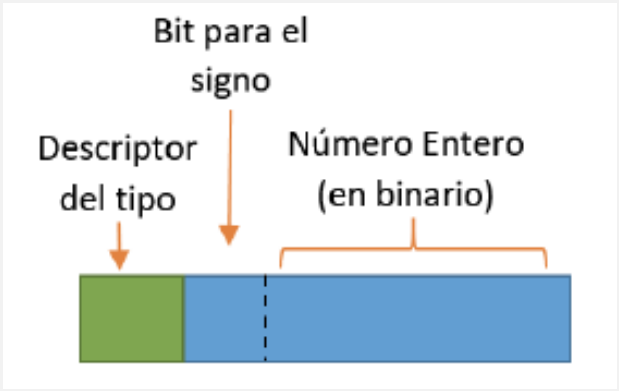
- Implementación: (continuación)

Con descriptor: en palabra separada o en la misma palabra.

Separada como en LISP



O en la misma palabra



ENTEROS (CONT.)

Java incluye 4 tamaños de enteros con signo que son: *byte, short, int y long*. Algunos lenguajes como *C++* y *C#* incluyen el tipo entero sin signo.

La mayoría de las computadoras actuales utilizan la notación llamada complemento a dos, para almacenar enteros negativos, la misma es conveniente para las operaciones de suma y resta.

SUBRANGOS

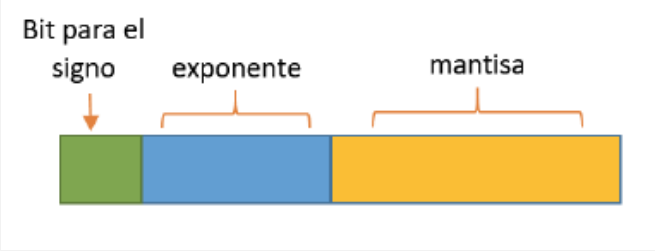
- Especificación: subtipo de los enteros en un rango restringido. Permite el mismo conjunto de operaciones que los enteros.
- Implementación: tienen dos efectos importantes sobre las implementaciones:
  Requerimientos de memoria reducidos: por ejemplo un entero en el intervalo 1..10 requiere solo 4 bits de almacenamiento para su representación.
  Mejor chequeo de tipos.

Tanto Ada como Pascal poseen el tipo subrango, en cambio, ni C ni Java lo admiten.

NÚMEROS REALES

De punto flotante

- Especificación: los valores también constituyen un subconjunto ordenado, con un máx y min. Además del tipo real otro atributo suele ser la precisión.
- Operaciones: las mismas que los enteros, más algunas propias de los reales.
- Implementación: usualmente la provista por el hardware. El número es dividido en una mantisa (números significantes) y un exponente, siguiendo el formato $N = 2^k \times m$. Por ejemplo: $-5 = -2^2 \times 1{,}25$.



NÚMEROS REALES

De punto fijo (específicos)

- Especificación: permiten especificar cantidad de dígitos antes y después del punto, son provistos por COBOL, C#.
- Operaciones: las mismas que los enteros, más algunas propias de los reales.
- Implementación: directamente por hardware o simulado por software.

ENUMERADOS

Se usan cuando queremos que una variable tome un pequeño número de valores simbólicos (sexo, estado civil, etc). La alternativa es representarlos con enteros, pero se pierde significado.

- Especificación: lista ordenada de valores distintos. El programador define los nombres literales y su ordenamiento.
- Operaciones: todas las relacionales, asignación, más las operaciones sucesor y predecesor.
- Implementación: se usan enteros, 0, 1, 2, ... Puede requerir menos bits que un entero común (es un subrango de valores siempre positivos).

Es decir se le asigna un valor entero a cada valor simbólico y se usan solo los bits suficientes para el intevalo de valores.

Algunos lenguajes que lo proveen: Pascal, Ada, C, C++, C#, Java.

BOOLEANO

Tipo de dato para representar **valores lógicos**: verdadero (true) y falso (false).

- Especificación: en Pascal y Ada son enumeraciones definidas por el lenguaje.
- Operaciones: las más comunes incluyen la asignación y los operadores *a*nd, or, not.
- Implementación: se requiere solamente un bit, pero se suele usar un byte o palabra, debido a que un único bit no puede ser direccionable en memoria.

CARACTER

- Especificación: tienen un caracter simple como valor. Usan codificaciones numéricas que corresponden a algún conjunto de caracteres estándard (ASCII, Unicode, etc).
- Operaciones: incluyen las operaciones relacionales, la asignación y a veces operaciones para probar si un valor caracter pertenece a las clases especiales (letra, dígito o caracter especial).
- Implementación: se guarda el código. Casi siempre son manejados directamente por el hardware y el sistema operativo subyacente, debido a su uso en entrada/salida.

STRINGS (CADENAS DE CARACTERES)

Objeto de datos **compuesto de una secuencias de caracteres**. Es considerado elemental aunque su implementación involucre una organización más compleja.

¿Todos los lenguajes soportan el tipo string?

Tipo primitivo: (Java, Pascal, Ada, JavaScript, Snobol4)

Arreglo de caracteres: (C, C++)

STRINGS (CADENAS DE CARACTERES)

- Especificación y sintaxis:

Longitud fija declarada:

Cobol → `S1 PIC X(5) VALUE "mundo"`
Java (Clase String)→ `String S2 = "Hola"`
Pascal → `S3 : string[6];`
`S3 := 'Hola mundo';` -sólo almacena *Hola m-*

Longitud variable (con límite declarado):

C y C++ → `char s4[100] = "Hola mundo";`

Longitud variable sin límite:

Java (Clases StringBuffer StringBuilder)
→ `StringBuilder S5 = new StringBuilder("Hola")`
Snobol4 → `S6 = 'Hola mundo'`

STRINGS (CADENAS DE CARACTERES)

- Operaciones:

- ✓ Asignación (copia)
- ✓ Concatenación
- ✓ Comparación (relacionales)
- ✓ Longitud
- ✓ Selección de substrings por posición o concordancia de patrones
- ✓ Conversión a números
- ✓ Selección, inserción y supresión de un caracter

STRINGS (CADENAS DE CARACTERES)

- Implementación:

Longitud es fija, se establece cuando se crea el objeto. Si no se utilizan todas las posiciones, se completan los espacios restantes:

P	R	E	P	R	O	C	E	S	A	M	I	E	N	T	O	“”	“”	“”	“”

STRINGS (CADENAS DE CARACTERES)

- Implementación (cont.):

Se reserva espacio para una longitud máxima de string, pero la cadena actual puede ser más corta.

Se debe incluir información que permita determinar cual es el último caracter del string actual. Alternativas:

Almacenar longitud actual y máxima:

16	20	P	R	E	P	R	O	C	E	S	A	M	I	E	N	T	O		

Almacenar delimitador de fin de string (como en C):

P	R	E	P	R	O	C	E	S	A	M	I	E	N	T	O	\0			

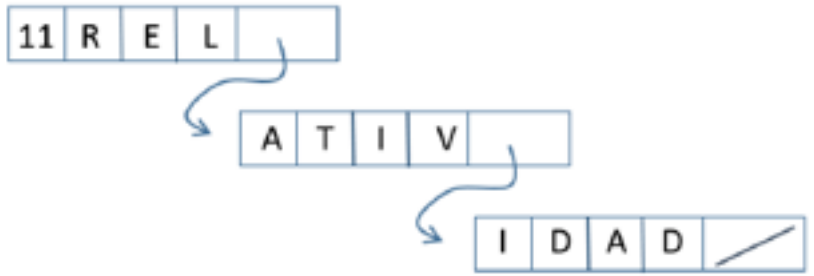
STRINGS (CADENAS DE CARACTERES)

- Implementación (cont.):

Longitud variable (sin límite): el string puede tener cualquier longitud, y variar dinámicamente por la inserción o supresión de caracteres.

Alternativas de implementación:

- Lista encadenada de caracteres. Fácil inserción y supresión de caracteres.



ó

- Arreglo contiguo de caracteres que contenga el string

PUNTEROS

Los punteros solucionan el requerimiento de tener objetos de datos de tamaño variable.
Los lenguajes de programación con tipos de datos punteros deben poseer:

- Un tipo de datos elemental puntero para almacenar **la dirección de otro objeto de datos**, o un valor especial NULL (o NIL).

- *Una operación de **creación** para objetos de datos de tamaño específico que retorna un bloque de memoria para el almacenamiento (en C: malloc o en C++/ADA/Pascal el new) y un **link** a ese espacio. Si no se provee la desasignación implícita también se requiere de una operación explícita (free en C y delete en C++). Estos OD pueden ser creados en cualquier punto de la ejecución del programa.*

- Una operación de **desreferenciación** para valores de punteros. En C *.

PUNTEROS

- Especificación:

Un tipo de datos puntero define una clase de OD, cuyos valores son direcciones de otros OD.

Un OD puntero se puede tratar de dos formas:

- *Los punteros sólo pueden hacer referencia a OD de un único tipo (Pascal, C y Ada).*
- *Los punteros pueden hacer referencia a OD de distinto tipo en diferentes momentos durante la ejecución del programa (Smalltalk).*

PUNTEROS

- Operaciones:

1. Operación de creación.
2. Operación de selección o desreferenciación.

- Implementación:

Un puntero es la dirección base del bloque de almacenamiento que representa al objeto de datos al que el puntero apunta.
Representaciones para los OD de tipo puntero:

1. Direcciones absolutas.
2. Direcciones relativas.

PUNTEROS

- *Direcciones absolutas:* un valor puntero se puede representar como la dirección de memoria real del bloque de almacenamiento para el OD.

- *Direcciones relativas:* un valor puntero se puede representar como un desplazamiento, respecto a la dirección base, de algún bloque de almacenamiento (del heap) más grande dentro del cual el objeto está asignado.

PUNTEROS

Direcciones Absolutas:

Ventajas:

1. Se puede asignar espacio para los objetos de datos en cualquier parte de la memoria.
2. La operación de selección es más efeciente, porque el valor del puntero mismo proporciona acceso directo al objeto de datos.

Desventaja: la administración de memoria es más difícil, dado que ningún objeto de datos se puede mover dentro de la memoria si existe un apuntador a él.

PUNTEROS

Direcciones Relativas:
Se requiere la asignación inicial de un bloque de almacenamiento, dentro del cual tiene lugar la asignación del objeto de dato puntero, por medio de la operación de creación.

Ventaja: se puede mover el bloque completo de memoria, en cualquier momento, sin invalidar ningún puntero.
Desventaja: la selección es más costosa porque se debe sumar el desplazamiento a la dirección base del área.

¿DUDAS?