

PROGRAMACIÓN PROCEDURAL

Unidad 2

OBJETOS DE DATOS TIPOS DE DATOS ELEMENTALES



Tema 2: Objetos de Datos – Tipos de Datos



Objetos de datos definidos por el programador:

Creados y manipulados por el programador a través de declaraciones y enunciados. Ej: variables, arreglos, etc.

Objetos de datos:

Representación de un elemento u objeto de la realidad de manera que pueda ser procesado por una computadora

Recipiente para almacenamiento y recuperar valores de datos

Objetos de datos definidos por el sistema:

*Se **crean automáticamente** sin intervención del programador*

Se construyen para mantenimiento durante la ejecución de un programa,

Pilas de almacenamiento
en tiempo de ejecución

Registros de
activación

Memoria
intermedia
de archivos

Objetos de Datos

Atributos

- **Identificador**
- **Tipo**
- **Valor**
- **Ubicación**

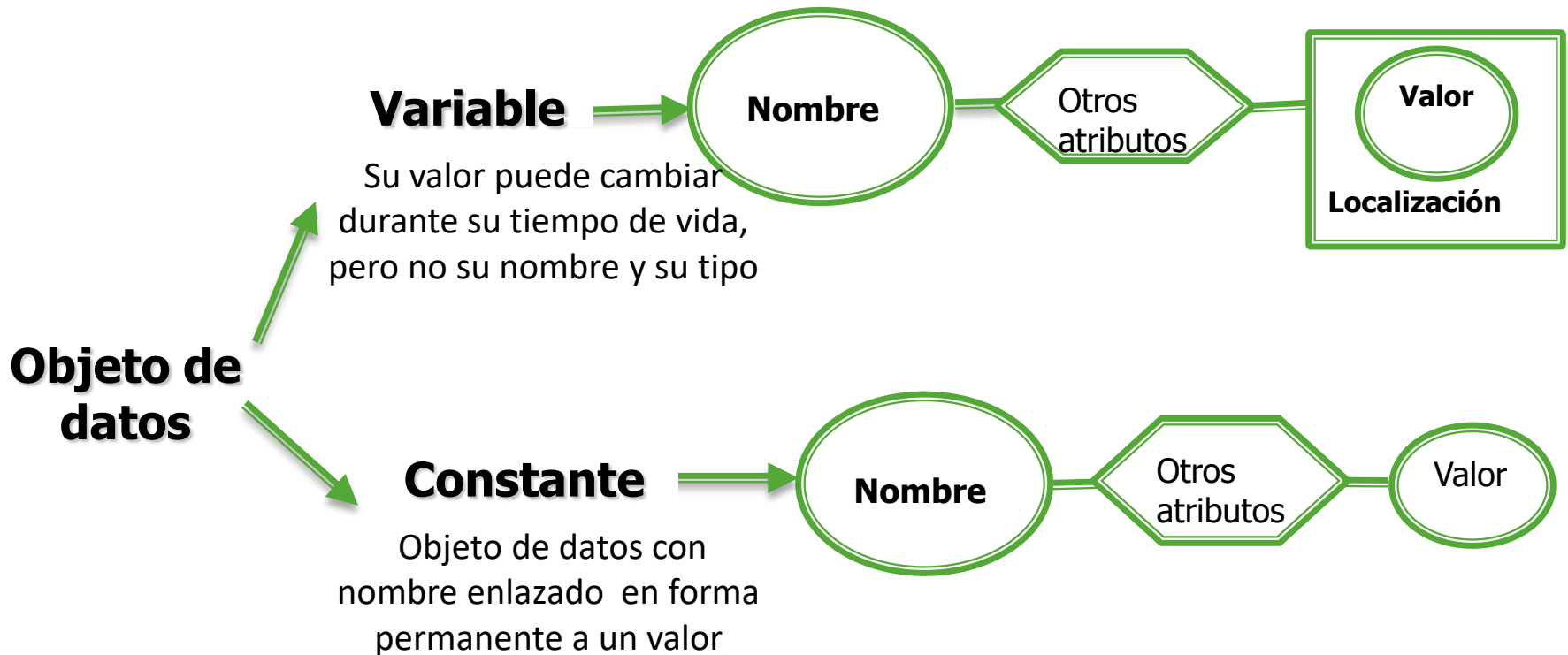
Clasificación

- **Elemental**
- **Estructurado**



```
main()
{
  int i, b=10, c[20];
  for(i=0; i<20; i++)
  {
    ....
  }
  ....
}
```

Objetos de Datos: Representación Formal



Objetos de Datos - Constantes

Lenguaje C

```
#include <stdio.h>
#define PI 3.1415926
int main()
{
    printf("Pi vale %f", PI);
    .....
    .....
}
```

Salida: Pi vale 3.1415926

El valor de PI se puede computar en tiempo de compilación. No necesita ocupar memoria.

Lenguaje C++

```
#include <stdio.h>
const int a = 5;
const int b = 2 + a;

main()
{ printf("El valor de b es %d", b);
    .....
    .....
}
```

Salida: El valor de b es 7

a, b son constantes cuyo valor se computa en tiempo de carga o al comienzo de la ejecución, por lo tanto su valor necesita ser almacenado en la memoria.

Tiempo de Vida

Tiempo durante el cual el objeto **puede usarse para** guardar valores de datos.

*Durante la ejecución
de un programa*

Algunos objetos existen desde el comienzo ó pueden crearse dinámicamente durante la ejecución



Algunos objetos persisten durante toda la ejecución ó se pueden destruir durante la misma.

Enlace o Ligadura



➤ Enlace o ligadura de un elemento de programa

Elección de una propiedad dentro de un conjunto de propiedades posibles.

➤ Tiempo de enlace

Momento del programa durante el cual se realiza el enlace

- Tiempo de definición del lenguaje
- Tiempo de implementación del lenguaje
- Tiempo de traducción
- Tiempo de ejecución

***Estáticas
o
tempranas***

Dinámicas

Ligaduras – Tipos

$$X = X + 10$$

- el conjunto de los posibles *tipos de datos para X* se fijan en **tiempo de definición del lenguaje**
- el conjunto de los posibles *valores* que puede tomar la variable X se determinan en **tiempo de implementación del lenguaje**
- el tipo de dato asociado a X elegido por el programador es un enlace realizado en **tiempo de traducción**
- Si la variable X es de un tipo definido por el programador (lo admite Pascal o C), el conjunto de sus posibles valores se fijan en **tiempo de compilación**. En lenguajes como Prolog y SmallTalk el programador no define tipo
- 10 se representa como una serie de bits en **tiempo de implementación**, mientras que su representación decimal en el programa se hace en **tiempo de definición del lenguaje**

Tipos de ligaduras

Eficiencia de ejecución



Enlaces estáticos
C , Pascal

Vs

Flexibilidad



Enlaces dinámicos
Lisp

El traductor debe conservar los enlaces de tal manera que se realice la operación apropiada durante la traducción y la ejecución



Tabla de Símbolos

Es una estructura de datos que contiene una entrada por cada identificador declarado encontrado en el programa fuente, con la siguiente información:

tipo de variable

tipo de parámetro formal

nombre del subprograma

entorno de referencia, etc.

Tipos de Datos

Un tipo de datos es una clase de objetos de datos ligados a un conjunto de operaciones para crearlos y manipularlos.

Primitivos

datos integrados al lenguaje

Definidos por el programador

el lenguaje provee recursos para definir nuevos tipos

Especificación

Atributos: distinguen objetos de ese tipo, ej . para tipo arreglo nombre dimensión, tipo de dato de componentes.

Valor: especifica valores que pueden tomar objetos de este tipo de dato.

Operaciones: refieren a distintas formas de manipular los objetos de ese tipo.

Implementación

Representación de almacenamiento: refiere a la forma que se representan en memoria.

Representación de algoritmos y procedimientos que definen las operaciones

Especificación de Tipos Elementales de Datos

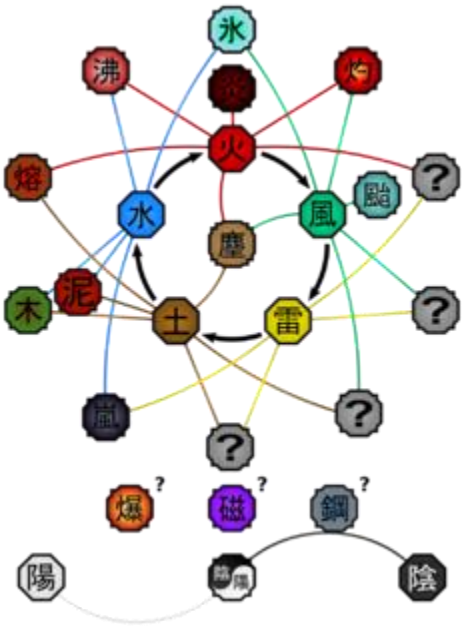
Es una clase de objeto de datos que **contiene un solo valor**

Atributos: se mantienen invariables durante su tiempo de vida.

Valor: generalmente ordenado, existe un máximo, un mínimo y dados dos valores distintos del dato, uno es mayor que el otro.

■ Operaciones

- **Primitivas:** se especifican como parte de la definición del lenguaje.
- **Definidas por el programador:** procedimientos que el programador genera.



Implementación de tipos elementales de datos

Representación de almacenamiento en memoria

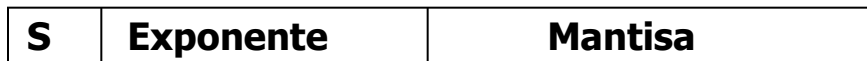
La representación de almacenamiento para **valores enteros o reales** generalmente utiliza la representación binaria que se usa en el hardware para esos valores



(sin
descriptor)

Entero

↑
Bit de
signo



norma 754 de IEEE (establecida en 1985)

Real

8 bits para exponente
de 2. (-127 a 128)

Signo: 1 bit
0 + **1**
-

23 bits. Número normalizado: el primer bit siempre es 1, se puede omitir ya que es insertado automáticamente por hardware.

Implementación de tipos elementales de datos

Representación de algoritmos y procedimientos que definen operaciones

- a) Como operación de hardware
- b) Simulada por software a través de subprogramas
- c) Simulada por software como una secuencia de código de línea

a) y b) función

```
#include <math.h>
```

```
main()
```

```
{int a,b; float c,d;
```

```
int base, exponente, r;
```

```
printf("\n sumas: %d %f", a+b,c+d);
```

```
scanf("%d %d", &base, &exponente);
```

```
r=pow(base, exponente);
```

```
printf("\n Potencia: %d", r);
```

```
}
```

c) macro

```
#define maximo (a,b) a<b ?b:a
```

```
main()
```

```
{
```

```
int a,b,m;
```

```
printf("\n Ingrese dos valores distintos");
```

```
scanf("%d %d", &a, &b);
```

```
m=maximo(a, b);
```

```
printf("\n El mayor es : %d", m);
```


```
getchar();
```

```
}
```

Declaraciones

Una **declaración** es un enunciado del programador que sirve para comunicar al traductor información que **permite establecer ligaduras**.

Ventajas

- Informan al traductor acerca de la representación de almacenamiento para un objeto de datos
 - Permiten que el traductor, determine cual es la operación particular a la que hace referencia el operador, en el caso de operaciones polimórficas
 - Informan sobre el tiempo de vida de un objeto
 - Verificación estática de Tipos
- 

Declaraciones : Almacenamiento y tiempo de vida de objetos

Ejemplo:

```
double calculo (float a )
```

```
{  
return a * 1.20;  
}
```

- Representación de almacenamiento
- Operación a que hace referencia en caso de operadores homónimos
- Tiempo de vida de un objeto
- Verificación estática de Tipos

```
void main (void)
```

```
{  
float sueldo,s,s1;  
int na,b,c  
c=na+b;  
s=sueldo +2000;  
s1=calculo(sueldo);  
int *p;  
p=malloc(30 * sizeof(int)) ;  
.....  
free(p);  
}
```



Verificación de tipos



La verificación de tipos es el proceso que realiza el compilador o el intérprete para verificar que todas las construcciones en un programa tengan sentido en términos de los tipos de sus constantes, variables, procedimiento o cualquier otra entidad que involucre.

Verificación estática

Se realiza durante la **traducción** de un programa, con información proporcionada por el programador a través de las declaraciones.

Verificación dinámica

Se realiza durante la **ejecución**. Si la información de tipos se conserva y se verifica en tiempo de ejecución, la verificación es dinámica.

Verificación Estática ¿Cómo se realiza?

- El traductor recoge información (variables y operaciones) desde la TS la cual se va construyendo a partir de las de las declaraciones.
- **Verifica la validez de los argumentos** y determina el tipo de datos del resultado, información que guarda para verificar operaciones posteriores.
- **Si la operación es homonimia o polimórfica**, el nombre de la operación se puede reemplazar por el nombre de la operación específica que usan esos argumentos.
- Como no se realizan marcas de tipo en los objetos en tiempo de ejecución, **se gana almacenamiento y tiempo de ejecución.**

Si un lenguaje detecta en forma estática los errores de tipo de un programa, se dice **que es de tipo fuerte o fuertemente tipificado**

Verificación dinámica de tipos ¿Cómo se realiza?

- Para implementarla se guarda una **marca de tipo en cada objeto (descriptor)**, que indica el tipo de datos asociado al mismo. Cuando se realiza la verificación, lo primero que se efectúa es la verificación de marcas de tipo de argumentos.
- La operación se realiza si los tipos de argumentos son correctos, de lo contrario se indica un error.
- Para cada operación, se anexan las marcas de tipo a sus resultados, para verificación de las operaciones siguientes.

Los intérpretes llevan a cabo verificación dinámica.



Verificación de tipos en C

Los compiladores de C aplican **verificación estática** de tipos durante la traducción, muchas **inconsistencias en tipos no causan errores de compilación**, sino que son automáticamente eliminados, presentando o no un mensaje de advertencia .

Conversión de tipos

Conversión Implícita (coerción)

- Invocadas automáticamente en ciertos casos de discordancia.
- Principio básico que gobierna las coerciones es **no perder información**.



Conversión Explícita

- El programador llama a funciones integradas para realizar una conversión de tipos.
- En C, se puede forzar a una expresión a ser de un tipo específico, usando el operador unario **cast**.

Conversión de tipos en C

Conversión implícita (coerción):

extensión- restricción

```
void main(void)
```

```
{char c;
```

```
int i;
```

```
float f;
```

```
double d, result;
```

```
result = (c / i) + (f * d) - (f + i)
```

```
result = (c / i) + (f * d) - (f + i)
```

char int float double float int

int double float

double

double

int result ?? double

Conversión explícita

```
#include <stdio.h>
```

```
void main(void)
```

```
{
```

```
int x=7;
```

```
printf("\n %f", x/2);
```

```
.....
```

```
}
```

Salida:?? ?

```
void main(void)
```

```
{
```

```
int x=7;
```

```
printf("\n %f", (float) x/2);
```

```
.....
```

```
}
```

Salida: ??

Operaciones sobre enteros

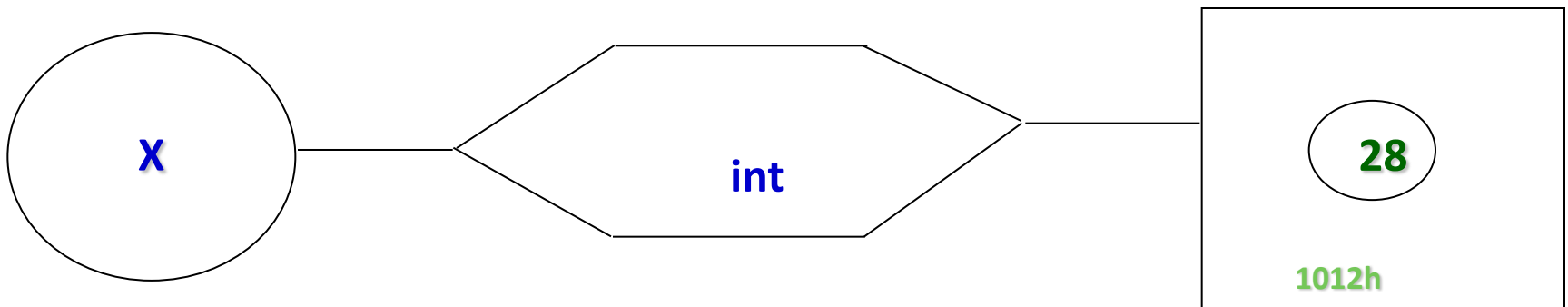
- Operaciones Aritméticas: +, -, *, /, ++, --
- Operaciones Relacionales <, >, <=, >=, ==, !=

Operación de asignación

Rvalue: hace referencia al valor que contiene una variable,

Lvalue : se refiere a la localidad de la variable

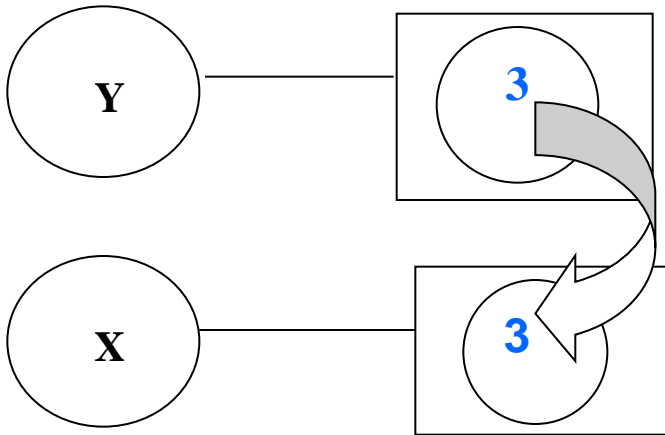
Para la asignación **X=28**,



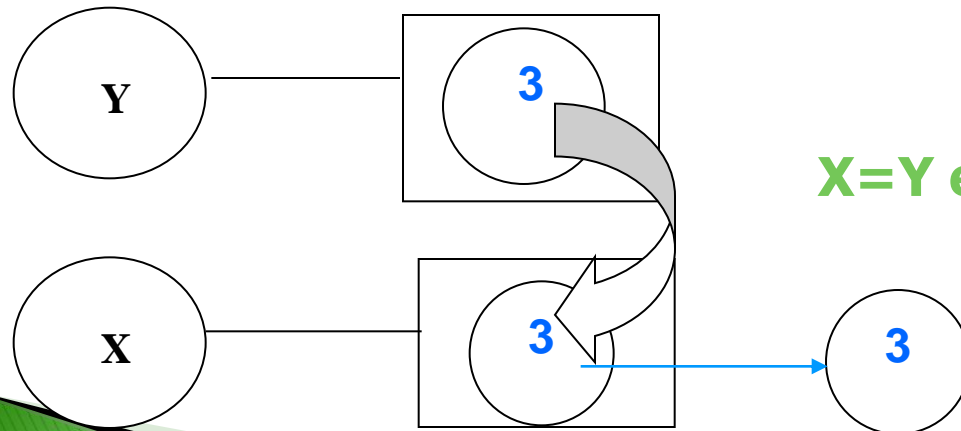
Valor R(x)= 28

Valor L(X)=1012h

Asignación – Distintas Implementaciones



$X := Y$ en lenguaje Pascal



$X = Y$ en lenguaje C

$X = Y + 3 * 22;$

- Computar el valor l de la variable X
- Computar el valor r de la expresión: $Y + 3 * 22$
 - 1 - computar el valor r de Y
 - 2 - computar el valor r de toda la expresión.
- Asignar el valor r calculado al objeto de datos del valor l computado.
- Devolver el valor r del objeto de datos asignado.

Particularidades de C

```
int a=10, b, c;
```

```
c=b=a*2;
```

Tipo de Dato Booleano

Especificación

Representa a objetos de datos que pueden tomar uno de dos valores posibles:

True o False.

En Pascal, el tipo booleano es tipo enumerado definido por el lenguaje, donde los valores simbólicos son TRUE y FALSE.

Operaciones :

Conjunción

Disyunción (inclusive)

Negación

Algunos lenguajes incluyen además: implicación, ó exclusivo.



Tipo de Dato Booleano

Implementación

Dos maneras de representación dentro de la unidad de almacenamiento direccionables (un byte o una palabra):

1- Se usa **un solo bit** del byte para representar con 0 o 1 y los restantes se ignoran.

(generalmente se usa el bit de signo)

2- Un valor 0 en **toda la unidad de almacenamiento** representa falso, cualquier otro valor representa verdadero.

Pascal: Utiliza un byte **False** se representa con 0

True se representa con 1

Lenguaje C, no se provee de tipos booleanos, usa objetos del tipo de dato entero para representar valores verdadero o falso.

0 representa valor falso

valor distinto de 0 representa valor verdadero

Tipo de Dato Carácter

Cadena ASCII

Especificación: toma como valor un solo carácter. El conjunto de posibles valores se corresponde con el conjunto de caracteres que maneja el hardware y el Sistema Operativo subyacente.

Conjunto de caracteres ordenados  orden alfabético entre cadenas

Implementación: Objetos de datos de tipo carácter manejados por el hardware y el S.O subyacente, debido a su uso en la entrada y salida de información.

En C, el tipo carácter es un subtipo del tipo entero. Cada carácter ocupa un byte y puede ser manipulado como un entero.

Caracteres ASCII de control

00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)
06	ACK	(reconocimiento)
07	BEL	(timbre)
08	BS	(retroceso)
09	HT	(tab horizontal)
10	LF	(nueva línea)
11	VT	(tab vertical)
12	FF	(nueva página)
13	CR	(retorno de carro)
14	SO	(desplaza afuera)
15	SI	(desplaza adentro)
16	DLE	(esc.vínculo datos)
17	DC1	(control disp. 1)
18	DC2	(control disp. 2)
19	DC3	(control disp. 3)
20	DC4	(control disp. 4)
21	NAK	(conf. negativa)
22	SYN	(inactividad sinc)
23	ETB	(fin bloque trans)
24	CAN	(cancelar)
25	EM	(fin del medio)
26	SUB	(sustitución)
27	ESC	(escape)
28	FS	(sep. archivos)
29	GS	(sep. grupos)
30	RS	(sep. registros)
31	US	(sep. unidades)
127	DEL	(suprimir)

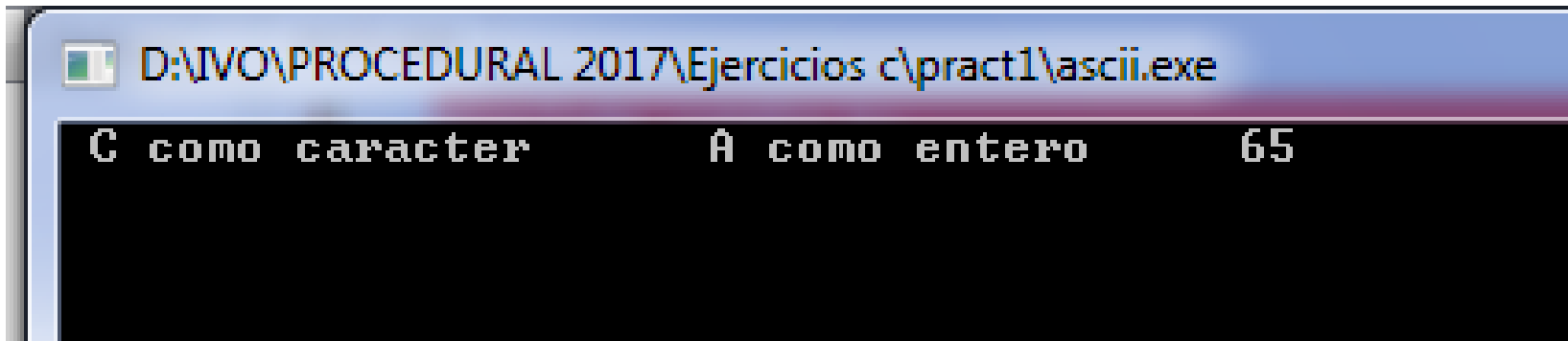
Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	\$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(72	H	104	h
41)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[123	{
60	<	92	\	124	
61	=	93]	125	}
62	>	94	^	126	~
63	?	95	_		

ASCII extendido (Página de código 437)

128	Ç	160	á	192	Ł	224	Ó
129	ü	161	í	193	ł	225	ô
130	é	162	ó	194	Ł	226	Ô
131	â	163	ú	195	ł	227	Ò
132	ä	164	ñ	196	—	228	ö
133	à	165	Ñ	197	†	229	Õ
134	â	166	ª	198	ã	230	µ
135	ç	167	º	199	Ä	231	þ
136	ê	168	¿	200	Ł	232	Þ
137	ë	169	®	201	Œ	233	Ú
138	è	170	™	202	ℒ	234	Û
139	ï	171	½	203	℥	235	Ü
140	î	172	¼	204	℥	236	ý
141	ì	173	¡	205	=	237	Ý
142	Ä	174	«	206	℥	238	—
143	Å	175	»	207	□	239	·
144	É	176	⋮	208	ð	240	≡
145	æ	177	⋮	209	Ð	241	±
146	Æ	178	⋮	210	Ê	242	—
147	ô	179		211	Ë	243	¾
148	ö	180	↓	212	È	244	¶
149	ò	181	À	213	Ì	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	Ã	215	Î	247	°
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	¶	217	Ɔ	249	°
154	Ü	186		218	Ɔ	250	·
155	ø	187	¶	219	■	251	°
156	£	188	¶	220	■	252	°
157	Ø	189	¢	221	⋮	253	°
158	×	190	¥	222	⋮	254	■
159	f	191	γ	223	■	255	nbsp

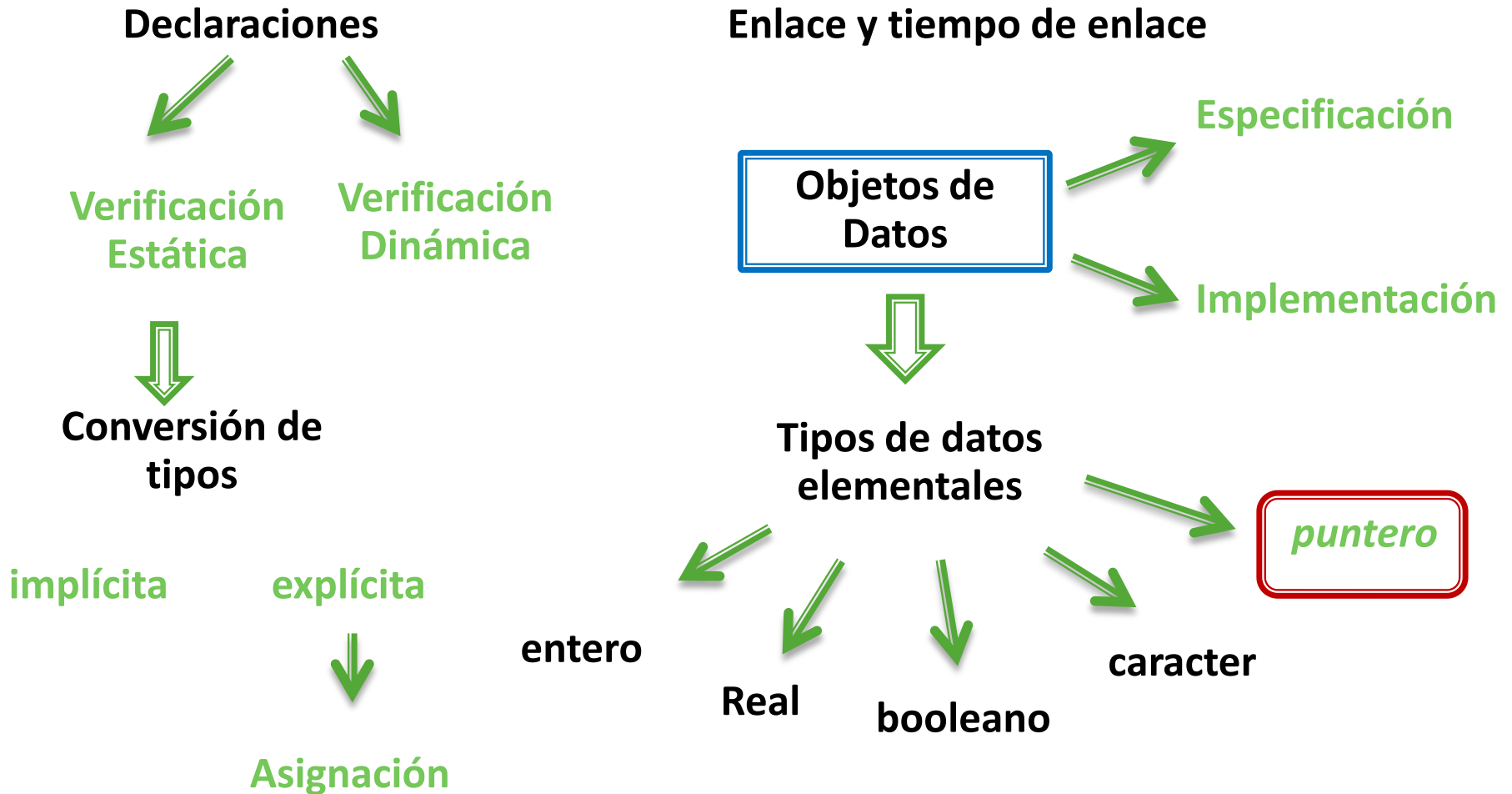
```
int main( )  
{  
    char c='A';  
    printf(" C como caracter %5c como entero %5d ",c,c );  
}
```



The screenshot shows a Windows command prompt window with the title bar "D:\IVO\PROCEDURAL 2017\Ejercicios c\pract1\ascii.exe". The command prompt displays the output of the program: "C como caracter A como entero 65". The text is formatted with spaces to align the character, the character name, and the integer value.

Output
C como caracter A como entero 65

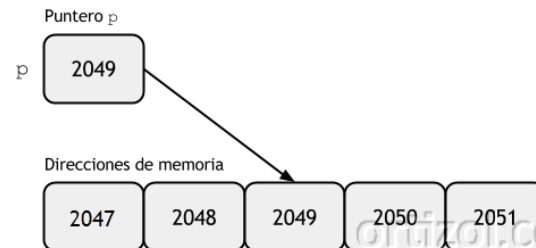
SINTESIS



Tipo de Dato Apuntador o Puntero

Un objeto del tipo apuntador, posee la dirección o localidad de otro objeto (valor L del objeto) o puede contener un apuntador nulo.

- No todos los lenguajes proveen este tipo de datos
- Tipos de datos elementales de lenguaje C para **gestión dinámica de memoria**



Permite:

- Al programador, en tiempo de ejecución crear variables y destruirlas cuando las considere innecesarias
- Crear estructuras de datos dinámicas (por ejemplo listas enlazadas).
- Devolver resultados a través de los parámetros de una función.

Punteros en C

```
main() {
```

```
int x, y, *p;
```

```
y=20;
```

```
x= y + 30;
```

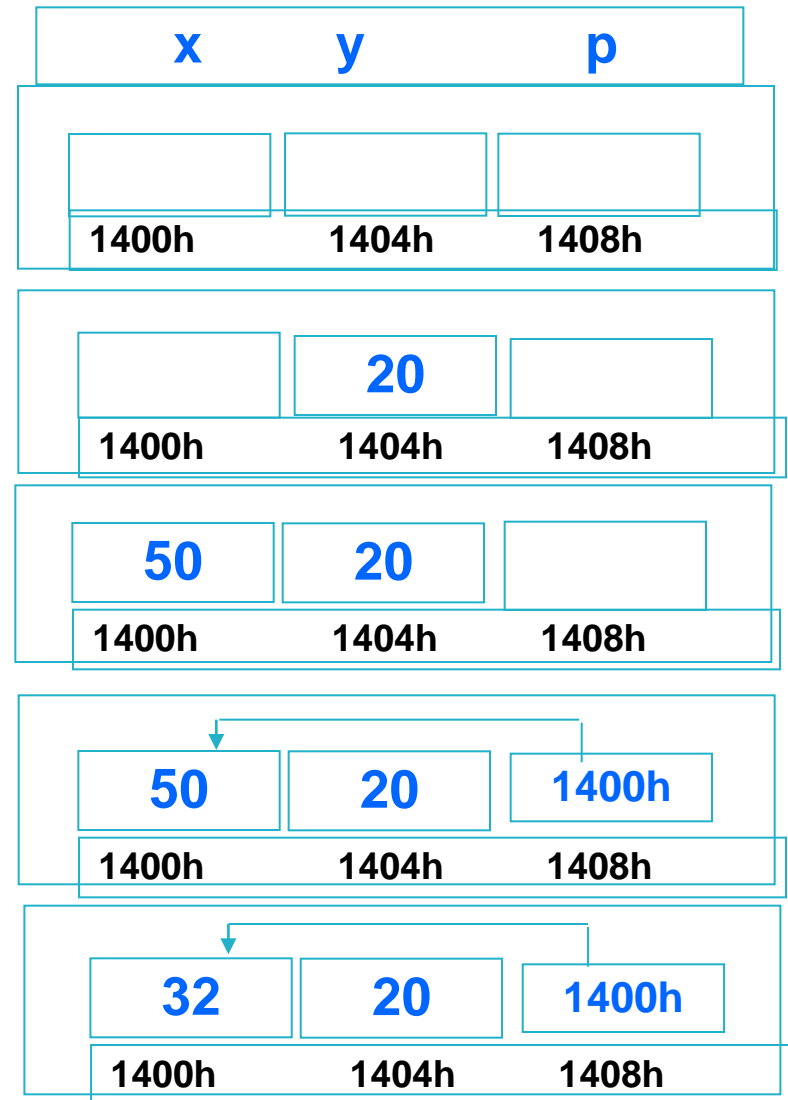
```
p=&x;
```

```
*p= y + 12;
```

```
print("%d %d", x, y);
```

```
}
```

Valor r(p)=valor l(x)



Punteros en C : Declaración y Operadores

Declaración de puntero: `< tipo > * < nombre variable >`

Operador de dirección (&): permite obtener la dirección de memoria de una variable.

`&x` expresa “la dirección de memoria de x”.

Operador de indirección o de contenido (*): permite obtener el contenido de la posición de memoria apuntada por un puntero,

`*p` expresa “el contenido de lo apuntado por p”



Punteros en C

ACTIVIDAD: Realizar el esquema memoria que permitirá visualizar los distintos pasos del algoritmo que se muestra:

```
#include <stdio.h>
void main (void )
{
    int a,b ,*p,*q;
    p=&a;
    *p=8;
    q=&b;
    *q=23;
    printf("%4d %4d \n",a, b);
    *p=*q+2;
    printf("%4d %4d %4d %4d \n",*p, *q, a , b);
    p=q;
    printf("%4d %4d %4d %4d\ n",*p, *q, a, b);
}
```



Punteros en C: Asignación de Valores

1. A través del operador &

```
char c , * punt;  
punt =&c;
```

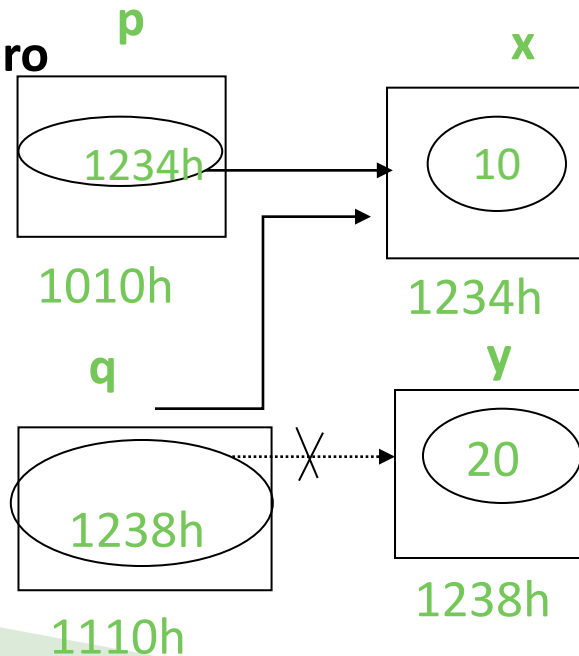
2. Con un dirección constante de memoria

```
int * punt =24A3;
```

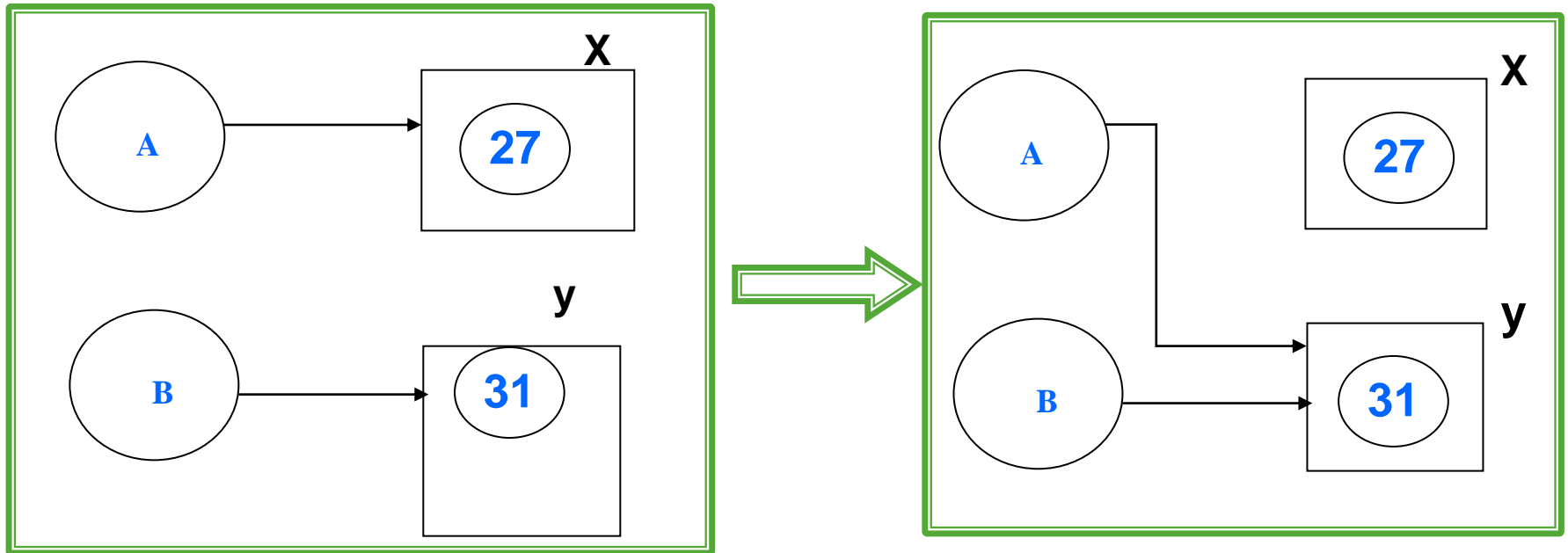
3. Por medio de NULL o void

4. Por medio de otro puntero

```
{  
int x=10, y=20,*p ,*q;  
  
q=&y;  
p=&x;  
  
q=p;  
}
```



Punteros en C – asignación de punteros



$A=B$ para A y B punteros.

Como B es un puntero, entonces el valor $r(B)$ es el valor l de algún otro objeto de datos.

Por lo tanto la asignación significa “hacer que el valor $r(A)$ se refiera al mismo objeto de datos que el valor $r(B)$ ”.