PROGRAMACIÓN DE SISTEMAS

Unidad 3 – Programación en Bajo Nivel

REPRESENTACIÓN DE DATOS EN MEMORIA

Representación de Datos en Memoria

- La memoria de las computadoras es **byte addressable**
- Cada dirección de memoria representa un byte
- Espacio de direcciones virtual (¿?)

- Tamaño palabra (word): tamaño nominal de enteros y punteros.

Representación de Datos en Memoria

- Notación hexadecimal

0xab12f1230

- Nota binaria

0b10101011000100101111000100100011000 0

- Notación decimal

45922325040

Representación de Datos en Memoria

- Tamaños de datos básico

C declaration	32-bit	64-bit
`char`	1	1
`short int`	2	2
`int`	4	4
`long int`	4	8
`long long int`	8	8
`char *`	4	8
`float`	4	4
`double`	8	8

Representación de Datos en Memoria

Orden en memoria

- **Big endian:** byte más significativo en dirección de memoria más baja
- **Little endian:** byte más significativo en dirección de memoria más alta

Precaución al transmitir datos por la red o al inspeccionar código ensamblador

Representación de Datos en Memoria

Machine	Value	Type	Bytes (hex)
Linux 32	12,345	`int`	39 30 00 00
Windows	12,345	`int`	39 30 00 00
Sun	12,345	`int`	00 00 30 39
Linux 64	12,345	`int`	39 30 00 00
Linux 32	12,345.0	`float`	00 e4 40 46
Windows	12,345.0	`float`	00 e4 40 46
Sun	12,345.0	`float`	46 40 e4 00
Linux 64	12,345.0	`float`	00 e4 40 46
Linux 32	`&ival`	`int *`	e4 f9 ff bf
Windows	`&ival`	`int *`	b4 cc 22 00
Sun	`&ival`	`int *`	ef ff fa 0c
Linux 64	`&ival`	`int *`	b8 11 e5 ff ff 7f 00 00

Representación de Datos en Memoria

Strings

- **ASCII**

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Representación de Datos en Memoria

Código

Simplemente es una secuencia de bytes

```
1   int sum(int x, int y) {
2       return x + y;
3   }
```

Linux 32:	55 89 e5 8b 45 0c 03 45 08 c9 c3
Windows:	55 89 e5 8b 45 0c 03 45 08 5d c3
Sun:	81 c3 e0 08 90 02 00 09
Linux 64:	55 48 89 e5 89 7d fc 89 75 f8 03 45 fc c9 c3

Representación de Datos en Memoria

Operaciones a nivel de bits

Operadores:

- &
- |
- ~

Representación de Datos en Memoria

Operaciones shift en C

Operadores

- >>
- <<

- Shift lógico vs shift aritmético
- En C, shift de enteros sin signo es lógico. Con signo, ambos pueden usarse (por lo general aritméticos)

Representando Enteros

Rangos

C data type	Minimum	Maximum
`char`	−128	127
`unsigned char`	0	255
`short [int]`	−32,768	32,767
`unsigned short [int]`	0	65,535
`int`	−2,147,483,648	2,147,483,647
`unsigned [int]`	0	4,294,967,295
`long [int]`	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
`unsigned long [int]`	0	18,446,744,073,709,551,615
`long long [int]`	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
`unsigned long long [int]`	0	18,446,744,073,709,551,615

Representando Enteros

Enteros sin signo

Si usamos un vector de bits de tamaño w para representar datos, podemos convertir a un número decimal sin signo así:

$$B2U_w(\vec{x}) \doteq \sum_{i=0}^{w-1} x_i 2^i$$

Cada número entre $0\ y\ 2^w - 1$ tiene un representación única.

Representando Enteros

Complemento de dos

Usado par representar número con signo:

$$B2T_w(\vec{x}) \doteq -x_{w-1}2^{w-1} + \sum_{i=0}^{w-2} x_i 2^i$$

Bit más significativo tiene peso **negativo.** Podemos reprentar numeros entre $-2^w \ hasta \ 2^w - 1$ (¿por qué?)

Representando Enteros

Convirtiendo entre números con signo y sin signo

```
short    int    v  = -12345;
unsigned short uv = (unsigned short) v;
printf("v = %d, uv = %u\n", v, uv);
```

v = -12345 y uv = 53191 (¿?). C mantiene la representación de bits (pero valores serán distintos de acuerdo al cast).

Representando Decimales

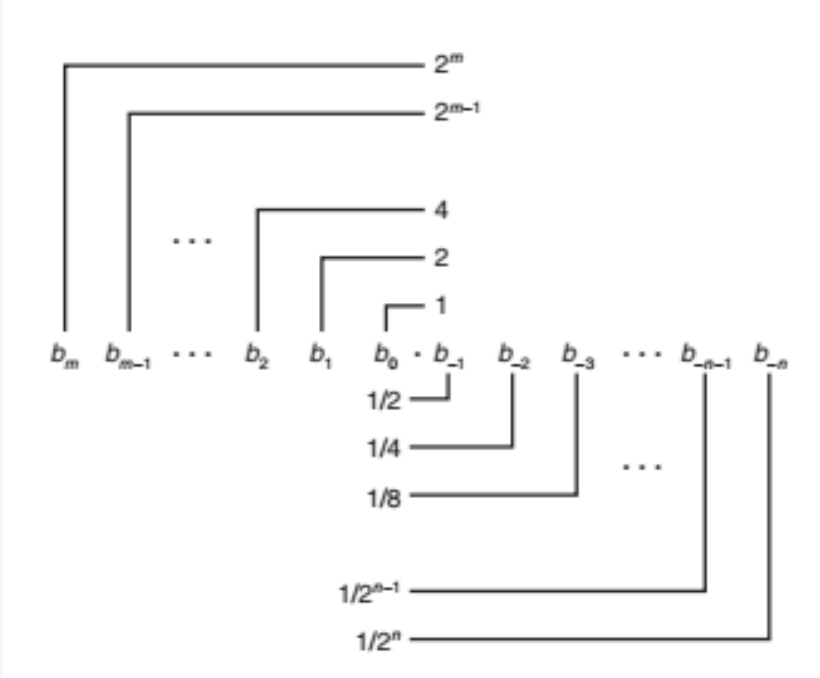
Numeros de Punto Flotante

Numeros fraccionales binarios:

$$b = \sum_{i=-n}^{m} 2^i \times b_i$$

Representación de Decimales

Numeros de Punto Flotante



Representación de Decimales

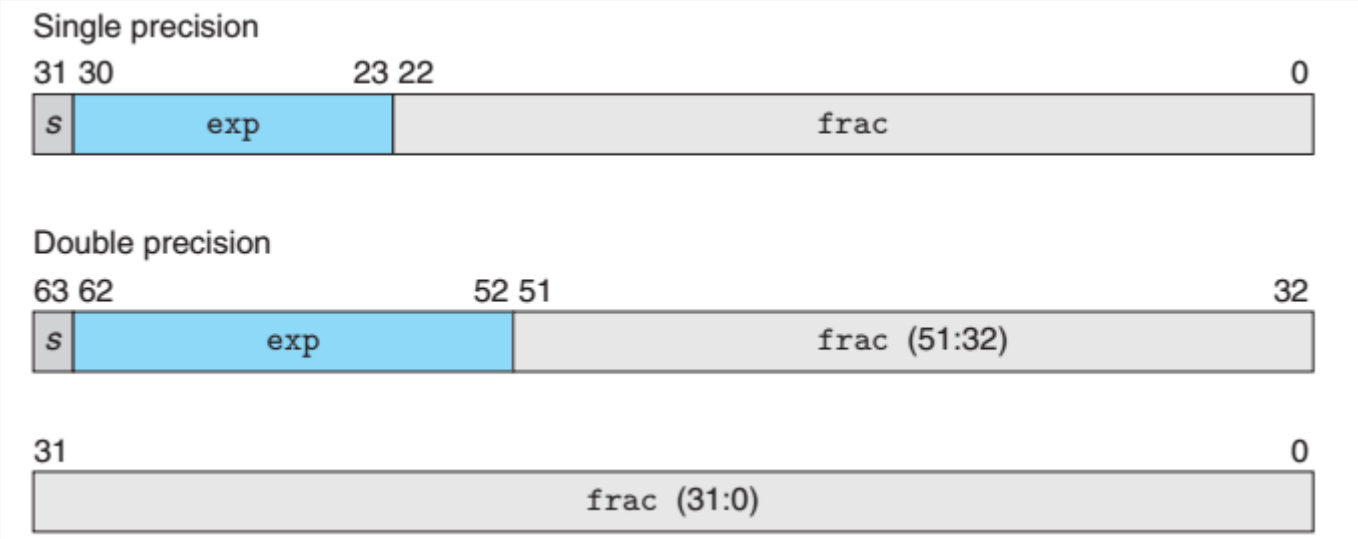
Representación IEEE de decimales

$$v = (-1)^s \cdot M \cdot 2^E$$

- S = signo (1 negativo, 0 positivo)
- M = significando, binario fraccional (entre 1 y 2 –e o entre 0 1 – e)
- Exponente potencia de 2

Representación de Decimales

Representación IEEE de decimales



Representación de Decimales

- Valor codificado depende de unos de los tres casos

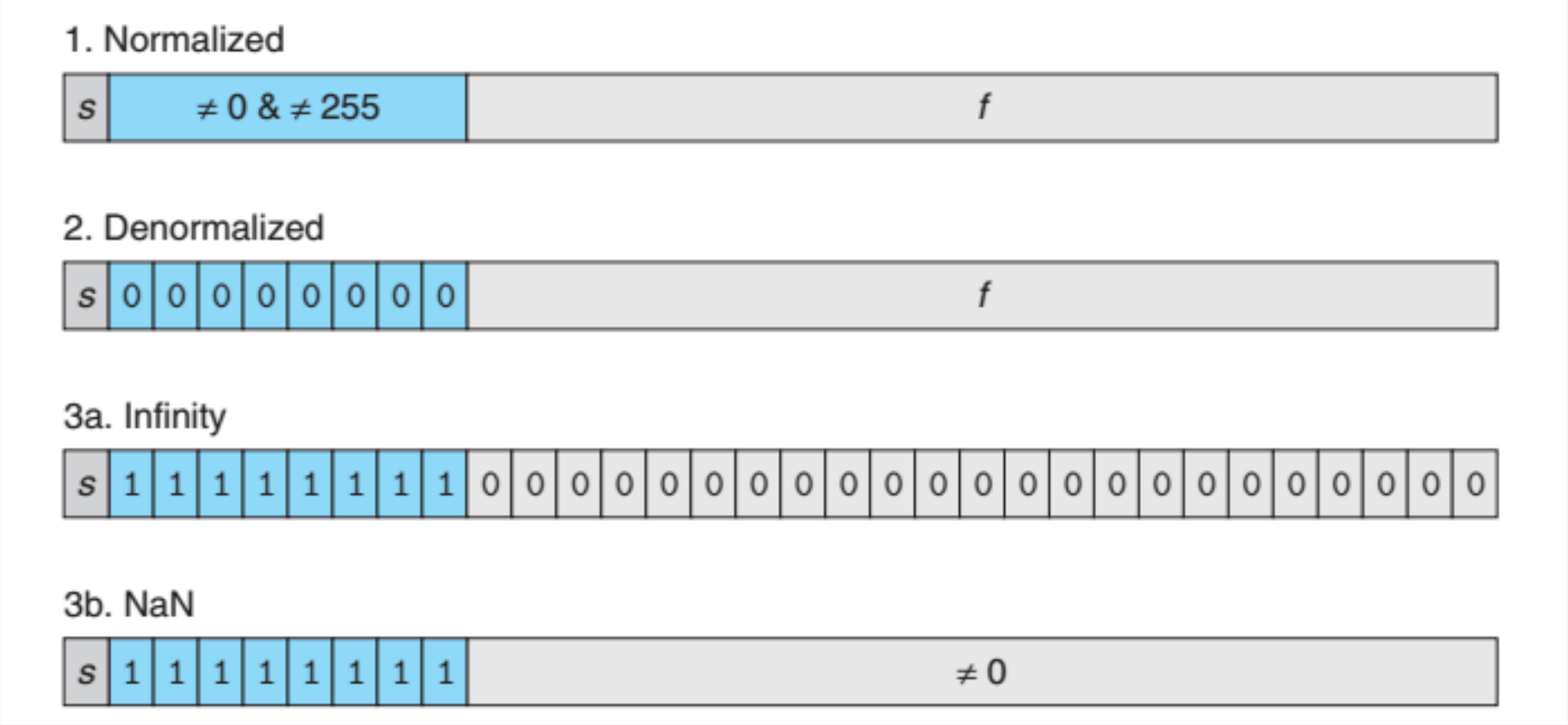
1. Valores normalizados (más comun)
   - *Cuando exponente no es todo ceros, ni todo unos*
2. Valor Desnormalizados
3. Valores especiales (fracción todo ceros → infinito u overflow NaN)

Representación de Decimales

Valores normalizados

- E = **exp** – sesgo (entero con sesgo). Sesgo es s^{k-1}(127 para precisión simple, 1023 para doble). Rangos de -127 a + 127 o -1022 a +1023.
- **frac** se interpreta como representación binaria de f = $0.f_{n-1}f_{n2}\ldots f_{0.}$ Entonces M = 1 + f.

Representación de Decimales



Description	Bit representation	Exponent			Fraction		Value		
		e	E	2^E	f	M	$2^E \times M$	V	Decimal
Zero	0 0000 000	0	-6	$\frac{1}{64}$	$\frac{0}{8}$	$\frac{0}{8}$	$\frac{0}{512}$	0	0.0
Smallest pos.	0 0000 001	0	-6	$\frac{1}{64}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{512}$	$\frac{1}{512}$	0.001953
	0 0000 010	0	-6	$\frac{1}{64}$	$\frac{2}{8}$	$\frac{2}{8}$	$\frac{2}{512}$	$\frac{1}{256}$	0.003906
	0 0000 011	0	-6	$\frac{1}{64}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{3}{512}$	$\frac{3}{512}$	0.005859
	⋮								
Largest denorm.	0 0000 111	0	-6	$\frac{1}{64}$	$\frac{7}{8}$	$\frac{7}{8}$	$\frac{7}{512}$	$\frac{7}{512}$	0.013672
Smallest norm.	0 0001 000	1	-6	$\frac{1}{64}$	$\frac{0}{8}$	$\frac{8}{8}$	$\frac{8}{512}$	$\frac{1}{64}$	0.015625
	0 0001 001	1	-6	$\frac{1}{64}$	$\frac{1}{8}$	$\frac{9}{8}$	$\frac{9}{512}$	$\frac{9}{512}$	0.017578
	⋮								
	0 0110 110	6	-1	$\frac{1}{2}$	$\frac{6}{8}$	$\frac{14}{8}$	$\frac{14}{16}$	$\frac{7}{8}$	0.875
	0 0110 111	6	-1	$\frac{1}{2}$	$\frac{7}{8}$	$\frac{15}{8}$	$\frac{15}{16}$	$\frac{15}{16}$	0.9375
One	0 0111 000	7	0	1	$\frac{0}{8}$	$\frac{8}{8}$	$\frac{8}{8}$	1	1.0
	0 0111 001	7	0	1	$\frac{1}{8}$	$\frac{9}{8}$	$\frac{9}{8}$	$\frac{9}{8}$	1.125
	0 0111 010	7	0	1	$\frac{2}{8}$	$\frac{10}{8}$	$\frac{10}{8}$	$\frac{5}{4}$	1.25
	⋮								
	0 1110 110	14	7	128	$\frac{6}{8}$	$\frac{14}{8}$	$\frac{1792}{8}$	224	224.0
Largest norm.	0 1110 111	14	7	128	$\frac{7}{8}$	$\frac{15}{8}$	$\frac{1920}{8}$	240	240.0
Infinity	0 1111 000	—	—	—	—	—	—	∞	—

PUNTEROS

Punteros

- Un puntero es una variables que contiene la **dirección** de otra variable.
- Para declararlo

```
int  *ptr;
```

- Cada puntero tiene asociado un tipo (¿Por qué?)

Punteros

¿Por qué?

- Permiten referenciar grandes estructuras de manera compacta
- Permite compartir información de manera fácil
- Permite reservar memoria durante ejecución

Punteros

- Operadores

* → de-referencia el puntero (permite obtener el valor al que apunta el puntero)

```
*p = 12;
```

& -> nos da la dirección de un objeto

```
int p = 43;
```

```
&p
```

Punteros

Asignando de puntero

```
int *prt;
prt = 40;
```

Asignando de valor

```
int *prt;
*prt = 40;
```

Puntero NULL → cuando queremos que puntero no haga referencia a datos validos (!)

Punteros

Pasando datos a funciones

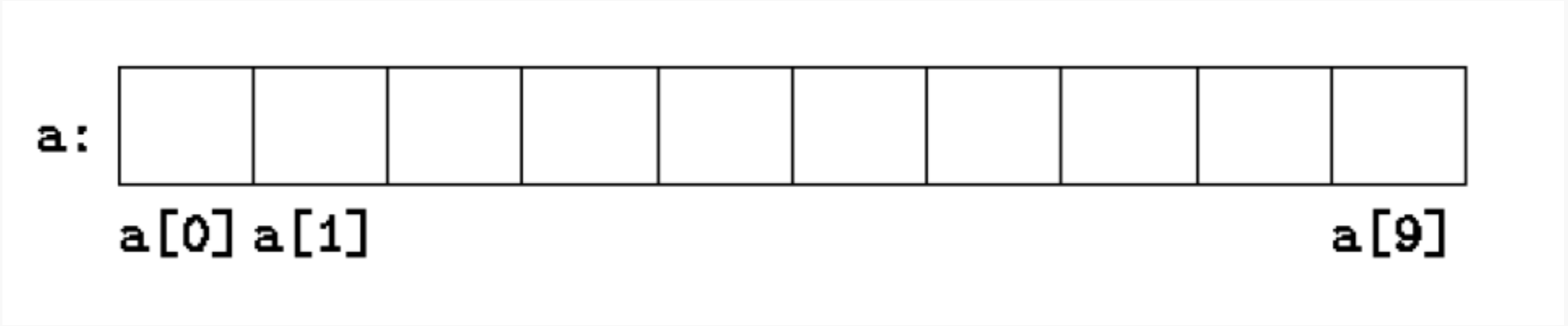
- Paso por valor
- Paso por referencia

Paso por referencia permite a funciones cambiar datos fuera del contexto de la función

Punteros

Punteros y arreglos

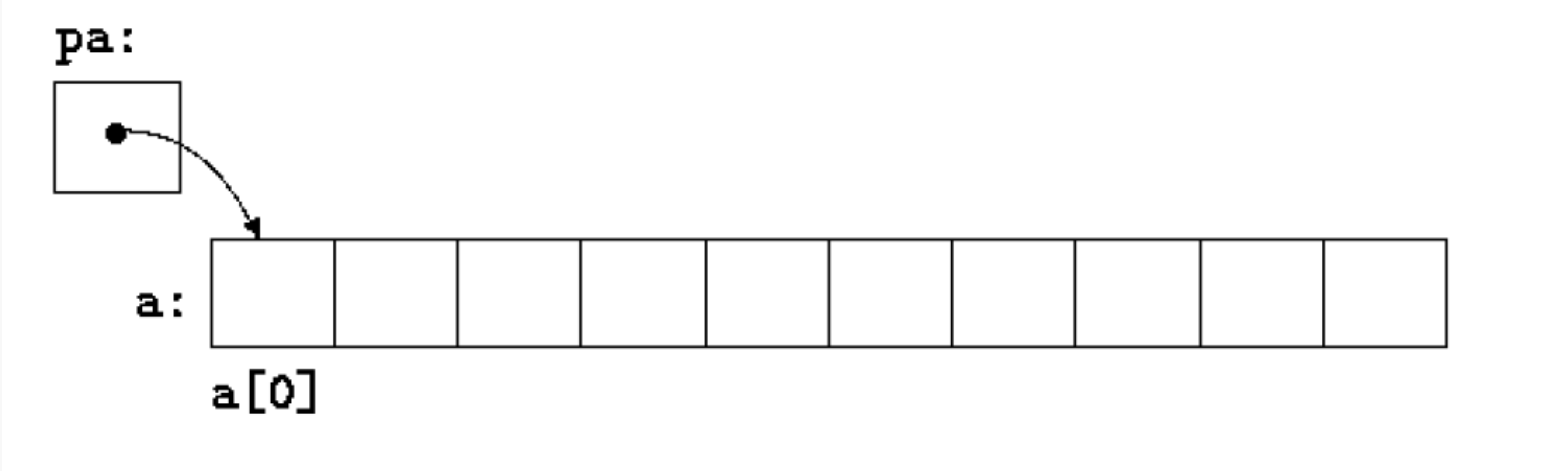
Fuerte relación entre los dos:



Punteros

int a[10];

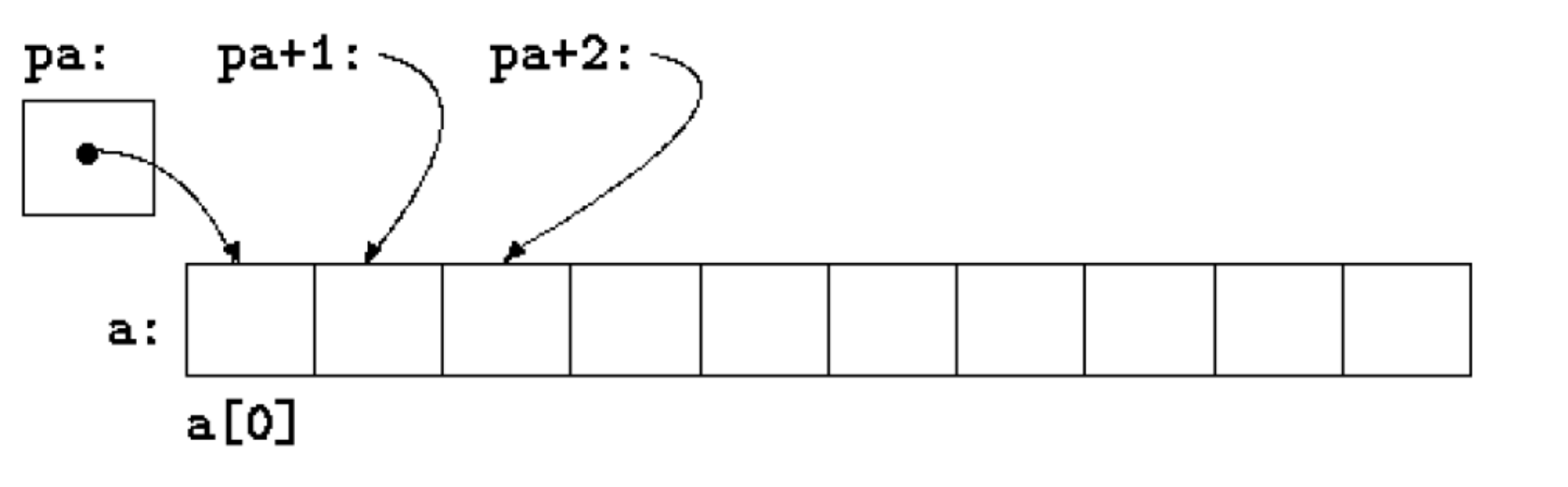
int *ptr = &a[0]



Punteros

Que pasa si hacemos:

*(pa+1) → ?



Punteros

Añadir uno al puntero es como avanzar un elemento en el arreglo **(no un byte)**

pa = *a[0]

Equivale a

pa = a

Entonces ***(pa+i)** equivale a **a[i].** Sin embargo a = pa y a++ so expresiones ilegales

Punteros

- Cuando pasamos un arreglo a una función, realmente estamos pasando la **dirección de memoria** del primer elemento del arreglo

```
void fn(int *arr){
        arr[3] = 40;
}
…
 int a[10] = 0;
 fn(a);
```

Punteros

- Podemos pasar parte del arreglo:

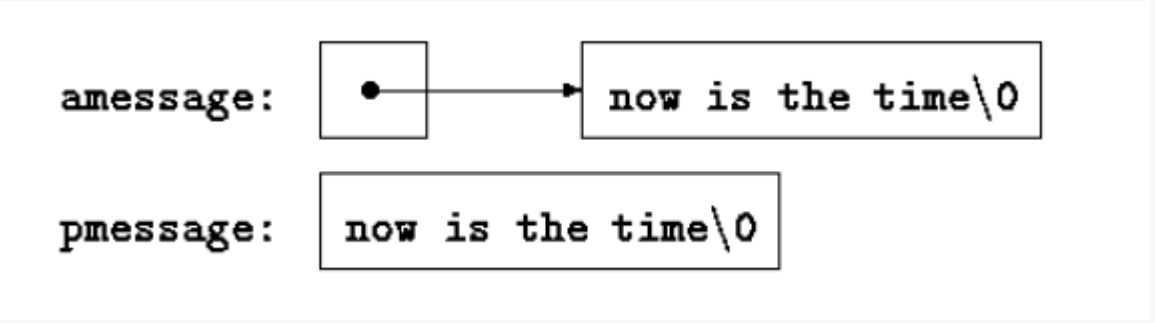
```
int a[10] = 0;

fn(a+2);      //pasamos el arreglo desde el elemento 2

fn(&a[2]);  //otra forma
```

Punteros

- Los punteros son una manera más conveniente para referirnos a strings:

char amessage[] = “now is the time” //podemos cambiar letras

char *pmessage = “now is the time” //string es inmutable aqui



Punteros

Paso por referencia

Punteros son ‘utiles para que funciones puedan cambiar valores fuera de su contexto

```
void f(int a){
        a = 10;
}
```

```
void f(int *a){
        *a=10;
}
```

Punteros

Arreglos de punteros y punteros dobles

char **ptr; //puntero doble

char *arreglo[10] //arreglo de 10 punteros del tipo char

Para inicializar arreglo de punteros:

arreglo = {“hola” , “chao” , … , “casa” };

Punteros

Arreglos Multidimensiones vs Arreglo de Punteros

int a[10][20] vs. int *b[10]

a tendra separada memoria para 200 ints + 10 punteros

b solo tendr’a asignado espacio para 10 punteros

Punteros

Punteros a funciones

Nos sirven para enviar funciones como argumentos

```
tipo (*nombre_funcion)(arg1, arg2, ...)
```

Punteros

Ejemplo:

```
float unafuncion(float a, float b){
        return a*b;
}
```

float (*mi funcion)(float, float) **//un puntero a funcion**

```
mi_funcion = unafuncion;
//Para usarla
float resultado = (*mi_funcion)(2.5, 3.4);
```

Punteros

Expresiones Complejas

```
char **argv
    argv:  pointer to char
int (*daytab)[13]
    daytab:  pointer to array[13] of int
int *daytab[13]
    daytab:  array[13] of pointer to int
void *comp()
    comp: function returning pointer to void
void (*comp)()
    comp: pointer to function returning void
char (*(*x())[])()
    x: function returning pointer to array[] of
    pointer to function returning char
char (*(*x[3])())[5]
    x: array[3] of pointer to function returning
    pointer to array[5] of char
```