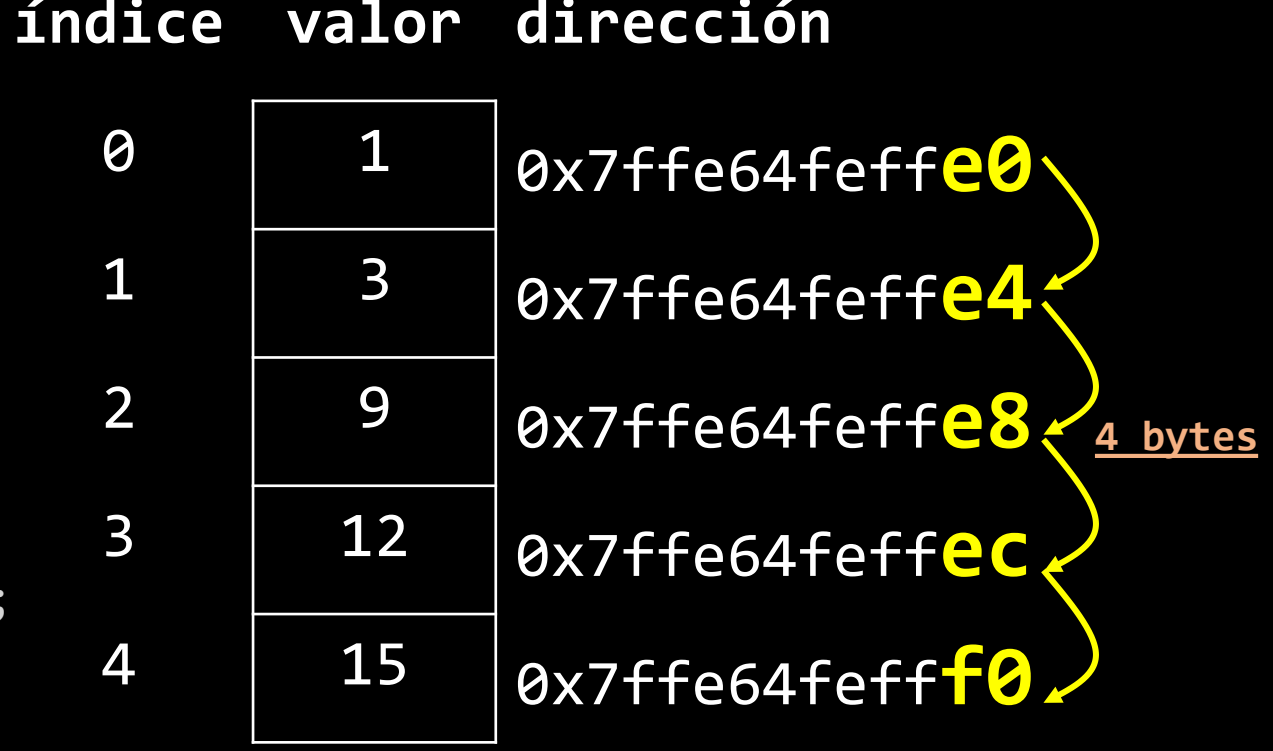
ARRAYS POINTERS

Oscar Paniagua @2023

array@punteros

```c
#define MAX 5

int main(){
    int arr[MAX] = { 1, 3, 9, 12, 15};
    int i;

    for(i=0;i<MAX;i++){
        printf("%d %d %p\n", i, arr[i], &arr[i]);
}
```



- ¿Por qué entre cada dirección hay 4 bytes? Si el array fuese del tipo char, cuanto habría?
- Porque es necesario que sean del mismo tipo?

array@punteros

```
int arr1[] = { 1, 3, 9, 12, 15};
```

índice	valor	dirección
0	1	0x7ffe64feff**e0**
	0	0x7ffe64feff**e1**
	0	0x7ffe64feff**e2**
	0	0x7ffe64feff**e3**
1	3	0x7ffe64feff**e4**
	0	0x7ffe64feff**e5**
	0	0x7ffe64feff**e6**
	0	0x7ffe64feff**e7**

```
char arr2[] = { 'a', 3, 'z', 12, 15};
```

índice	valor	dirección
0	'a'	0x7ffe64feff**e0**
1	3	0x7ffe64feff**e1**
2	'z'	0x7ffe64feff**e2**
3	12	0x7ffe64feff**e3**
4	15	0x7ffe64feff**e4**

array@punteros

```
int x[] = { 1, 3, 9, 12, 15};
```

x equivalente &x equivalente &x[0]

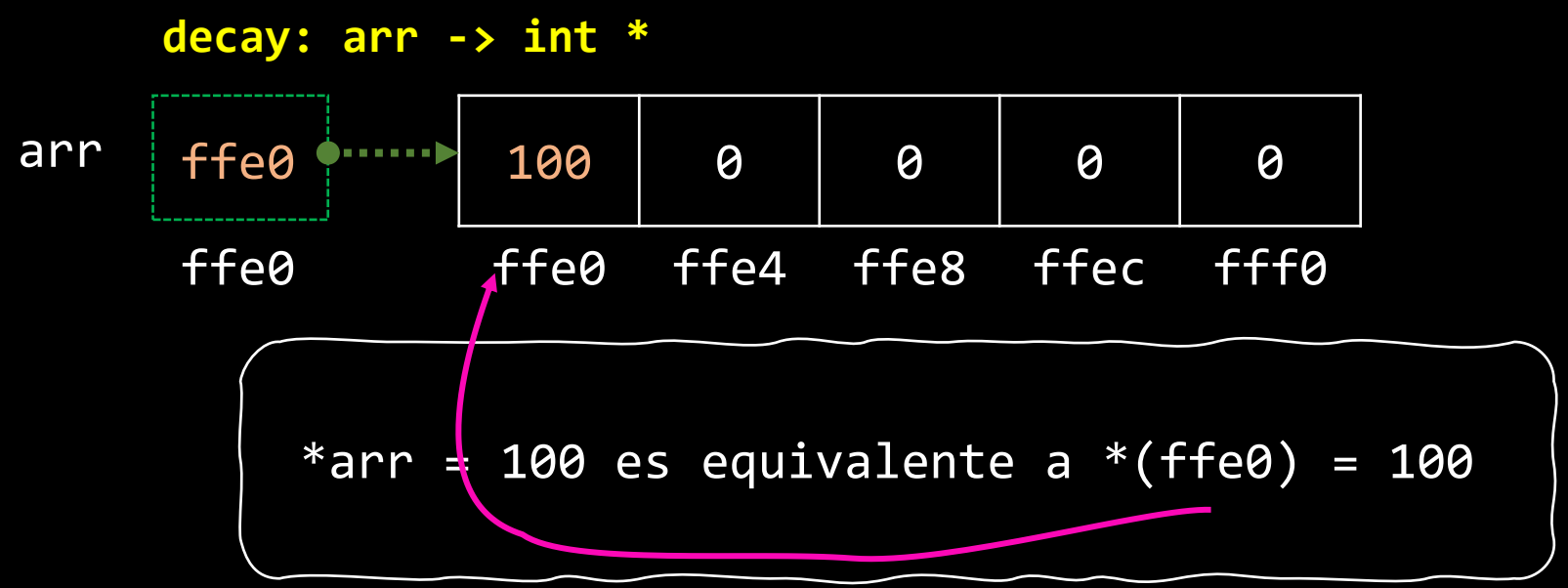
El nombre de la variable (identificador) representa la dirección de inicio del array (read-only), es decir, la dirección del primer elemento.

índice	valor	Dirección	
0	1	0x7ffe64feff**e0**	x (base address)
1	3	0x7ffe64feff**e4**	x + 4 bytes
2	9	0x7ffe64feff**e8**	x + 8 bytes
3	12	0x7ffe64feff**ec**	x + 12 bytes
4	15	0x7ffe64feff**f0**	x + 16 bytes

array@punteros

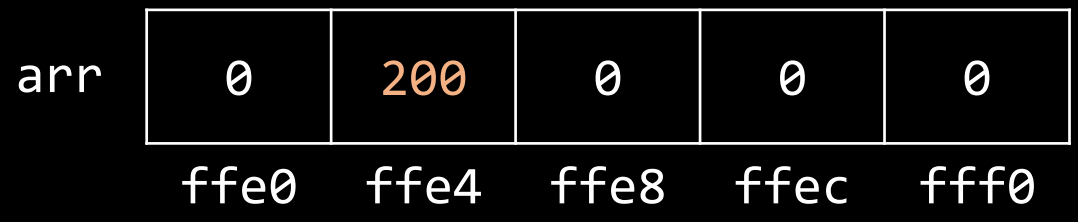
Podemos decir, que el nombre del array, puede ser utilizado como un puntero al primer elemento de un array (base address)

```c
int main()
{
    int arr[5] = {0};
    arr[0] = 100;
    *arr = 100;
    printf("%d\n", arr[0]);
    return 0;
}
```



array@punteros

```c
int main()
{
    int arr[5] = {0};
    *(arr+1) = 200;
    printf("%d\n", arr[1]);
    return 0;
}
```




```
*(arr+1) = 200 es equivalente a ...
*(ffe0 + 1x4) = 200
*(ffe4) = 200
```

array@punteros

```
int x[] = { 1, 3, 9, 12, 15};
```

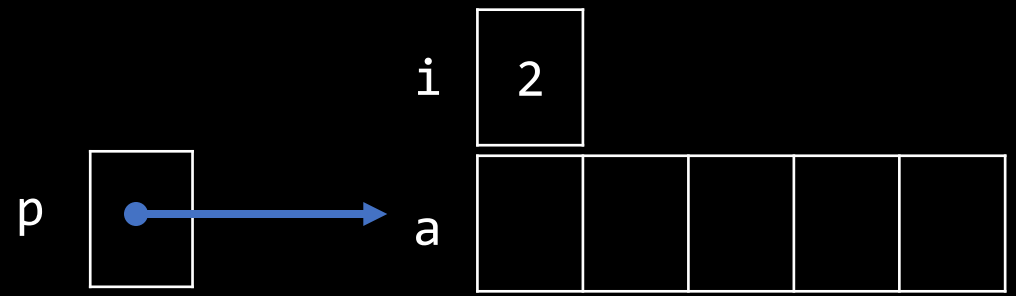
```
printf("%d %d %p\n", i, x[i], &x[i]);

printf("%d %d %p\n", i, *(x+i), (x+i));
```

índice	Valor	Dirección		Notación array	Notación puntero
0	1	0x7ffe64feff**e0**	x	x[0]	*(x)
1	3	0x7ffe64feff**e4**	x + 4	x[1]	*(x+1)
2	9	0x7ffe64feff**e8**	x + 8	x[2]	*(x+2)
3	12	0x7ffe64feff**ec**	x + 12	x[3]	*(x+2)
4	15	0x7ffe64feff**f0**	x + 16	x[4]	*(x+3)

4 bytes
Artimetica de punteros

array@ dualidad puntero-array

```
int i = 2;
char a[4], *p;
p = a;
```



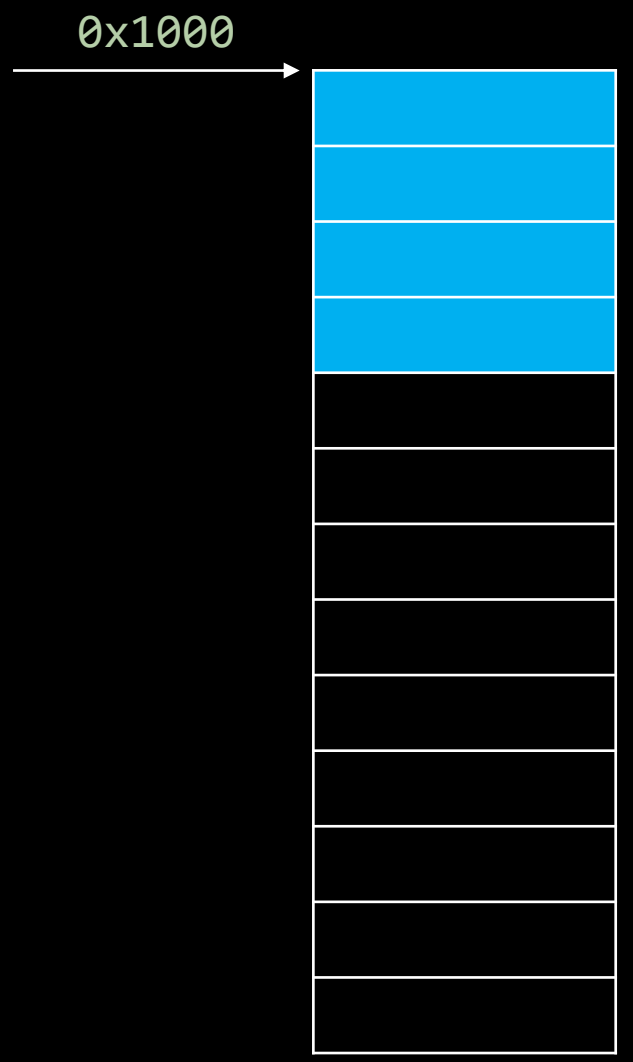

```
a[i]    <->    *(p + i)
p[i]    <->    *(a + i)
```

Tanto “a” como “p” apuntan a la misma dirección.
Esto se suele denominar dualidad arreglo puntero.

array@ dualidad puntero-array

```
(unsigned int*) 0x1000[0];
```

- En ocasiones especialmente al trabajar en sistemas embebidos sin sistema operativo base, se necesita acceder a registros mapeados en memoria, se conoce de antemano la dirección de memoria la cual es fija. Entonces se puede definir un puntero con un valor específico: #define registro base ((unsigned int*) 0x1000)
- Es decir, definimos una constante llamada registro base, cuyo valor inicial es 0x1000)



array@ dualidad puntero-array

```
(unsigned int*) 0x1000[1];
```

- En ocasiones especialmente al trabajar en sistemas embebidos sin sistema operativo base, se necesita acceder a registros mapeados en memoria, se conoce de antemano la dirección de memoria la cual es fija. Entonces se puede definir un puntero con un valor específico: #define registro base ((unsigned int*) 0x1000)
- Es decir, definimos una constante llamada registro base, cuyo valor inicial es 0x1000)

array@ dualidad puntero-array

```
#define registro_base ((unsigned int *)0x1000)

registro_base[0] = 1;
```

array@ dualidad puntero-array

```
int val;
char *a = (char*) 0x1000;
*a[100] = 20;
val = *(a + 100);
printf("%d", val);
```

SI SE PUEDE!
(algo similar realizaremos con memoria dinámica)

- aunque en nuestra PC generaría Segmentation fault (Linux trabaja en modo protegido, y al ejecutar nuestro programa solo nos deja acceder a la zona de memoria que nos asignó*)
- muy común al trabajar con uC donde se tiene acceso al mapa de memoria