

Titolo unità didattica: Strutture dati: array

[07]

Titolo modulo : Array in C

[07-C]

Generalità sulle proprietà del tipo strutturato array in C

Argomenti trattati:

- ✓ proprietà degli array C
- ✓ array 1D e 2D in C
- ✓ rappresentazione di array C
- ✓ array e puntatori in C
- ✓ notazione standard e notazione a puntatore
- ✓ passaggio di array a function C

Prerequisiti richiesti: AP-03-04-T, AP-07-01-T

array 1D

dichiarazione

```
<tipo> <nome_array>[<size>];
```

```
float inflaz_mese[12];  
int temperatura_oraria[24];  
char rigo[80];  
int psi[13];
```

<size> **deve** essere una **costante** o una **espressione costante**

<size> **non può** essere una **variabile**

array 1D

dichiarazione

```
<tipo> <nome_array>[<size>;
```

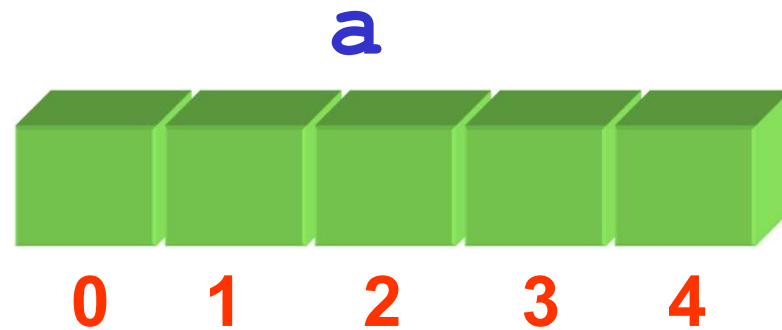
```
float inflaz_mese[12];  
int temperatura_oraria[24];  
char rigo[80];  
int psi[13];
```

in C gli array sono **allocati staticamente**

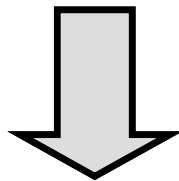
il compilatore C associa a un array uno spazio di memoria che dipende dal **size**

**tale spazio di memoria non può variare
(durante l'esecuzione del programma)**

```
int a[5];
```



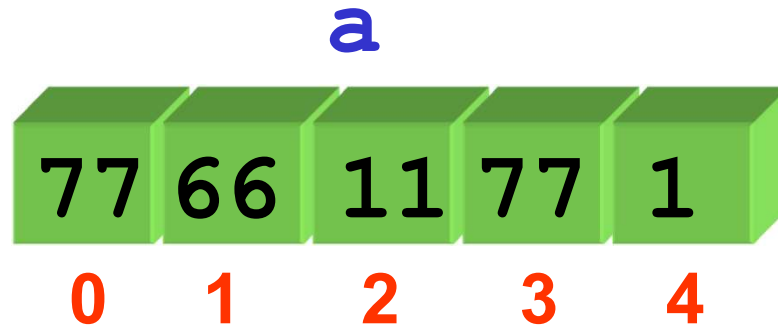
gli indici di un array C vanno sempre
da **0** a **size-1**



il **primo** valore dell'indice è sempre **0**
l'**ultimo** valore dell'indice è sempre **size-1**

```
int a[5];
```

gli indici di un array C vanno sempre da 0 a `size-1`



```
a[0] = 77;  
a[3] = a[0];  
a[2] = 11;  
a[4] = a[3] - a[2];  
a[1] = a[5] + a[2];
```

a[5] non esiste

array 1D

```
int a[5], n;  
n = 5;
```

```
for (i=0; i<n; i++)  
    scanf ("%d", &a[i]);
```

immissione da
tastiera di valori di
un array

```
for (i=0; i<n; i++)  
    printf ("%d ", a[i]);
```

visualizzazione dei
valori di un array

le operazioni su array C sono **solo**
componente per componente

non sono ammesse operazioni che
agiscono **globalmente** su un intero array

array 1D

dichiarazione-inizializzazione

```
int a[5]={22,-4,9,11,-6};
```

```
int a[]={22,-4,9,11,-6};
```

all'array viene associato lo spazio di memoria per memorizzare 5 dati di tipo `int`
cioè `5*sizeof(int)` byte

```
int a[5];  
a = {22,-4,9,11,-6};
```

array 2D

dichiarazione

```
<tipo> <nome_array> [<righe>] [<col>] ;
```

```
float matrice[5][5];  
int tabella[15][10];  
char pagina[40][80];
```

```
matrice[0][0] = 1.0;  
tabella[10][9] = tabella[2][1];  
pagina[3][7] = 'p';
```


array 2D

dichiarazione-inizializzazione

```
int matrice[2][3]= {{21,16,14},  
                    {12,22,30}};  
  
int A[3][2]= {{31,55},  
              {21,45},  
              {72,40}};
```

21	16	14
12	22	30

matrice

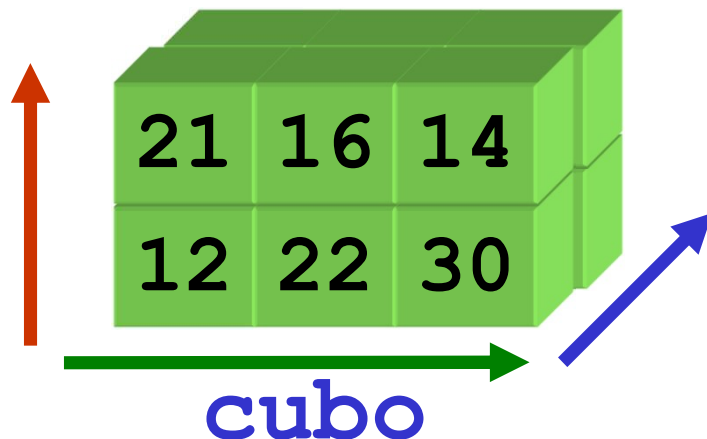
A

31	55
21	45
72	40

array *n*D

dichiarazione-inizializzazione

```
int cubo[2][3][2] = { { {21, 16, 14},  
                        {12, 22, 30} },  
                      { {-1, -6, 11},  
                        {91, 96, 94} } } ;  
  
cubo[1][1][1] = 0 ;
```



rappresentazione di array C

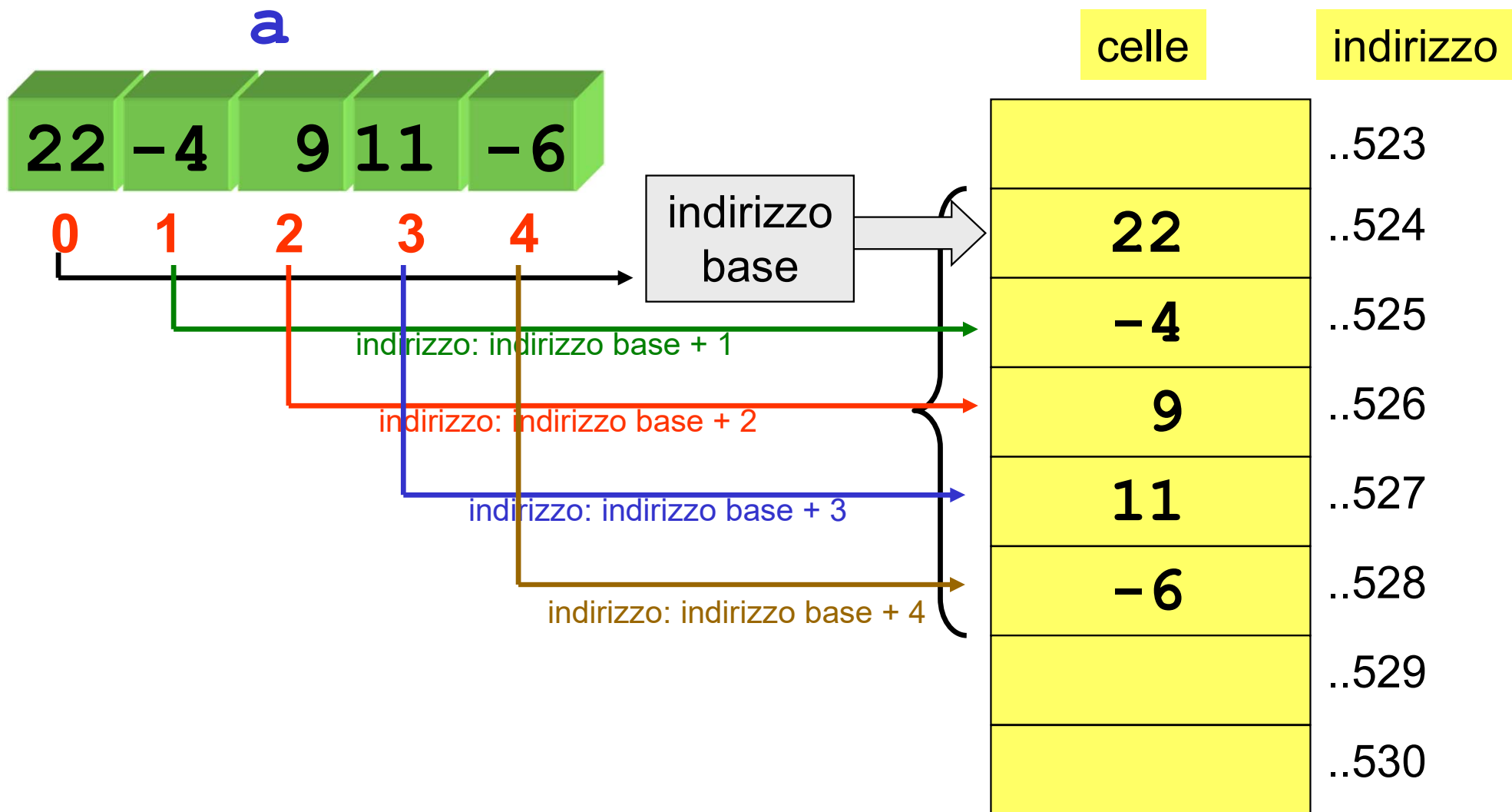
a un array vengono associate celle **contigue** di memoria (con indirizzi consecutivi)

l'**indirizzo** di un array è l'indirizzo della prima cella (**indirizzo base** dell'array)

il **primo** elemento dell'array (elemento di indice **0**) è memorizzato nella **prima** cella,
il **secondo** elemento dell'array (elemento di indice **1**) è memorizzato nella **seconda** cella,
.....

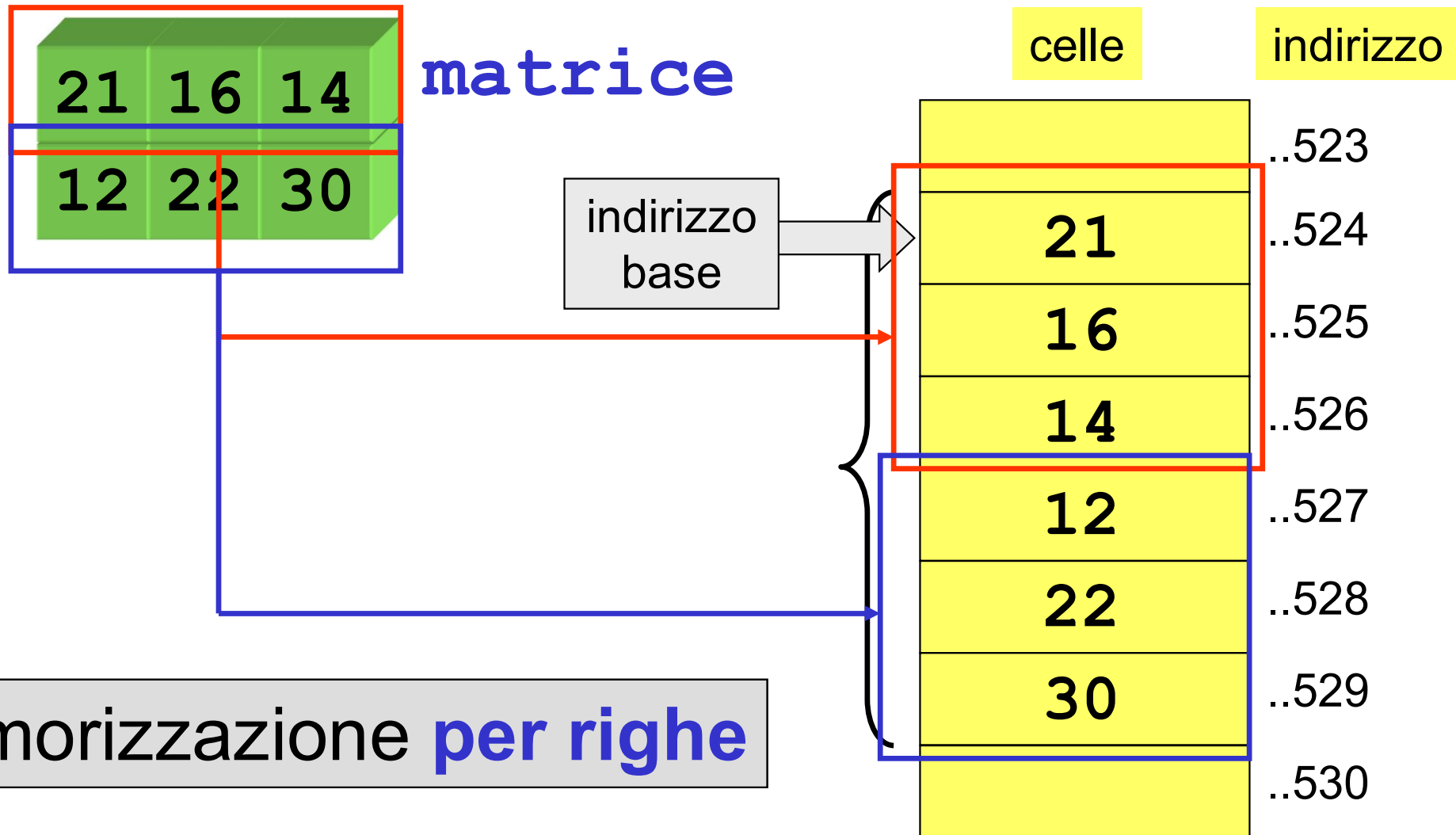
rappresentazione di array 1D

```
int a[5]={22,-4,9,11,-6};
```



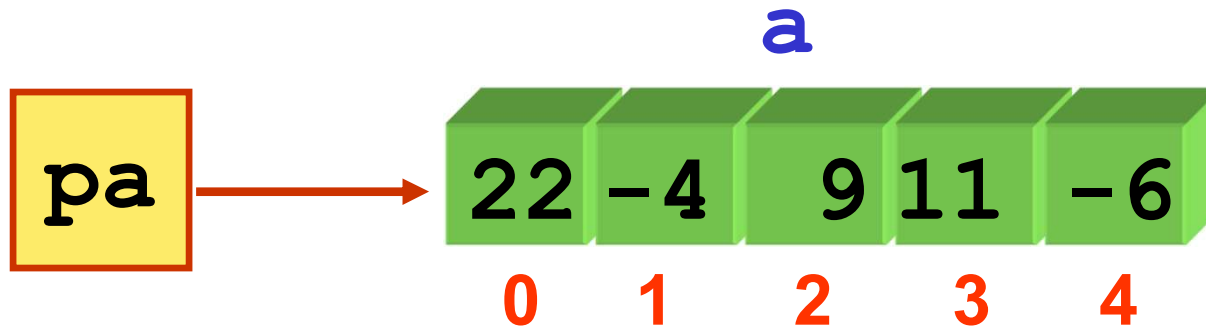
rappresentazione di array 2D

```
int matrice[2][3]= {{21,16,14},  
                    {12,22,30}};
```



array e puntatori

puntatore esplicito a un array



```
int a[5];
```

```
int *pa;
```

```
pa = &a[0];
```

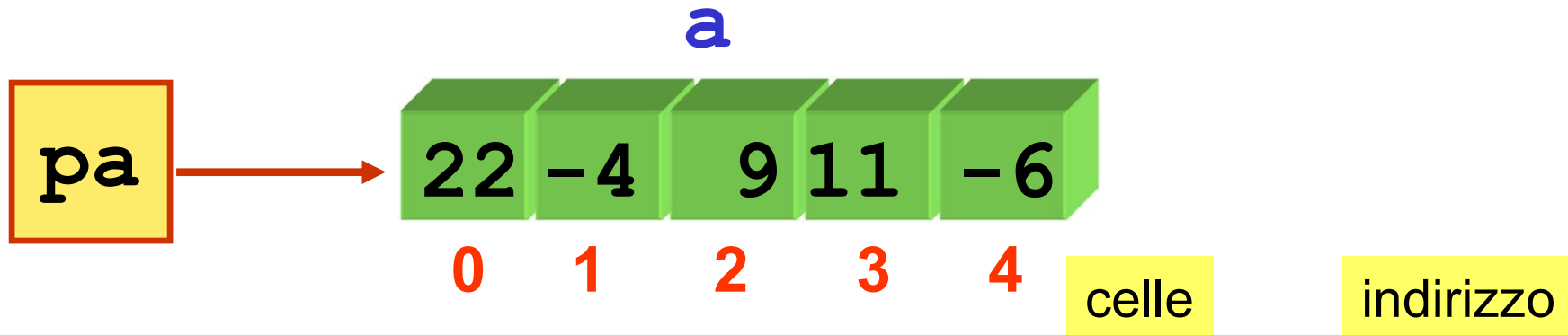
```
x = *pa;
```

`pa` contiene l'indirizzo
del primo elemento di `a`

`x` contiene il valore di
`a[0]`

array e puntatori

puntatore esplicito a un array



```
int a[5];  
int *pa;  
pa = &a[0];  
x = *pa;
```

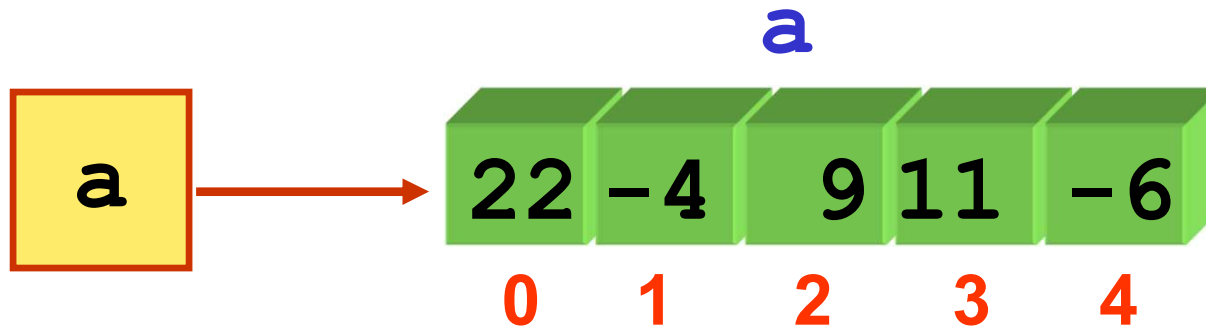
```
* (pa+1)  
* (pa+2)  
* (pa+3)
```

```
a[1]  
a[2]  
a[3]
```

aritmetica degli
indirizzi



array e puntatori



```
int a[5];
```

```
int *pa;
```

```
pa = a;
```

```
x = *pa;
```

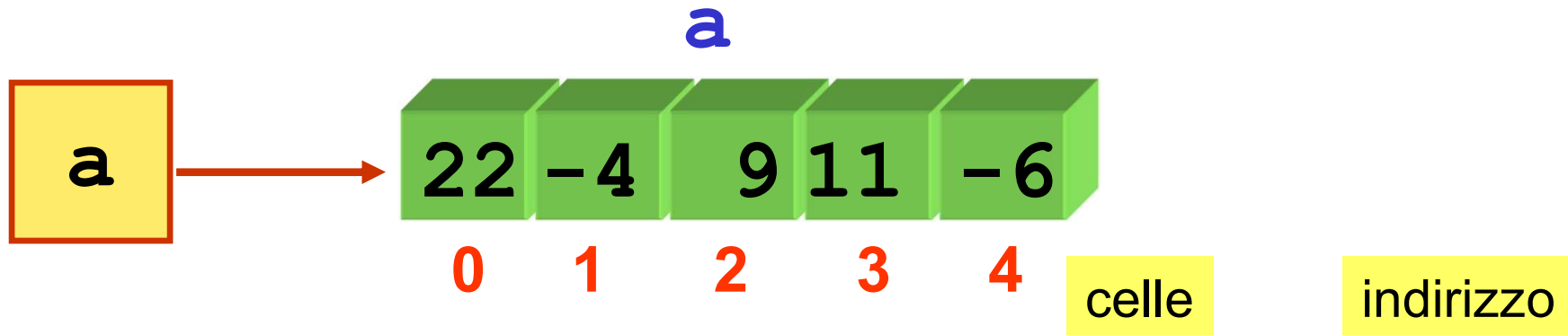
`pa` contiene l'indirizzo
del primo elemento di `a`

`x` contiene il valore di
`a[0]`

il nome di un array è un **PUNTATORE** (costante)
al suo **primo elemento**
(INDIRIZZO BASE dell'array)

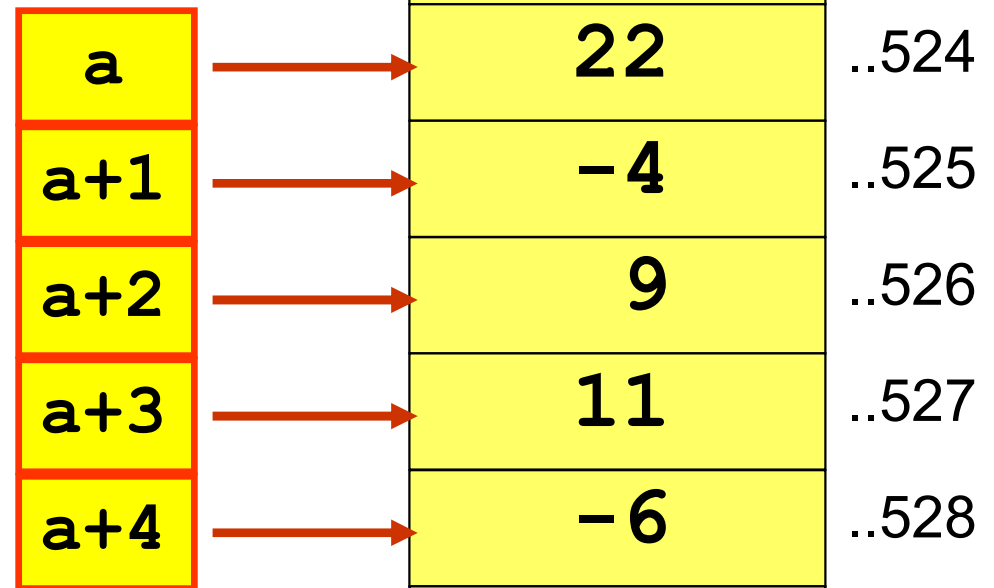
array e puntatori

il nome di un array è un **PUNTATORE**
(costante) al suo **primo elemento**
(INDIRIZZO BASE dell'array)



```
int a[5];  
int *pa;  
pa = a;  
x = *a; a[0]
```

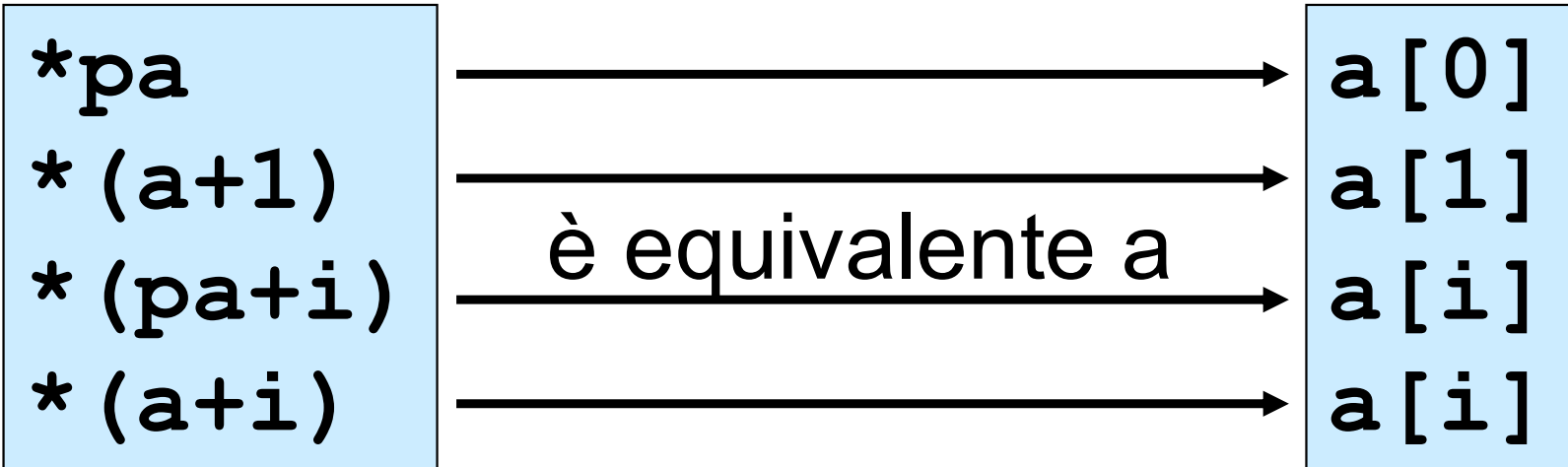
* (a+1)	a[1]
* (a+2)	a[2]
* (a+3)	a[3]



aritmetica degli
indirizzi

un elemento di un array può essere denotato
attraverso un **indice** (modalità **standard**)
oppure attraverso un **puntatore**
(il nome dell'array o un puntatore esplicito)

```
int a[5]= {14,22,11,6,21};  
int *pa; pa = &a[0];
```



- un elemento di un array può essere denotato
- ✓ attraverso un **indice** (notazione **standard**)
 - ✓ dereferenziando una espressione basata sull'aritmetica degli indirizzi (notazione a **puntatore**)

```
a[0]  
a[1]  
a[i]  
a[i+1]  
a[2*i]
```

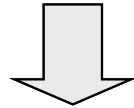
notazione standard

```
*a  
*(a+1)  
*(a+i)  
*(a+i+1)  
*(a+2*i)
```

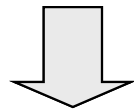
notazione a puntatore

passaggio di array a function

il **nome** di un array è un **puntatore** all'
indirizzo base dell'array



il passaggio di un **array** come parametro di
una function è **per riferimento (per indirizzo)**
e non **per valore**



non c'è bisogno di usare un puntatore esplicito all'array

basta usare il nome dell'array, sia quando è
argomento/parametro di input, sia quando è
argomento/parametro di output

passaggio di array a function

notazione standard

array come argomento (nella chiamata alla function):

✓ usare solo il nome dell'array

array come parametro (nell'intestazione di function):

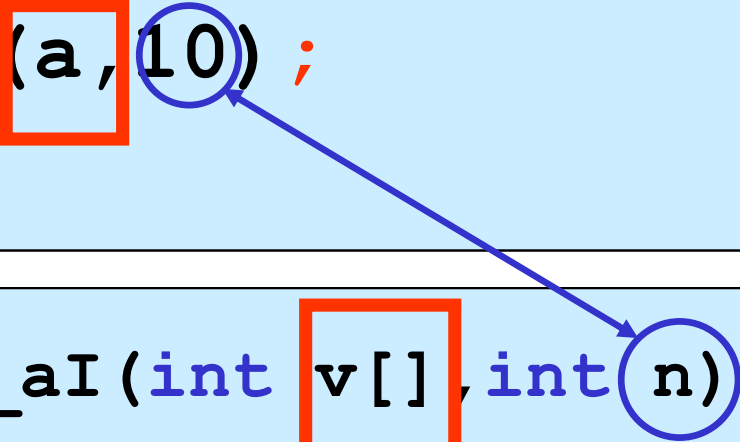
✓ usare un nome di array seguito da **[]**

passaggio di array a function

Esempio

notazione standard

```
...  
int a[10]={10,20,30,40,50,60,70,80,90,100};  
visualizza_aI(a,10);  
...
```



```
void visualizza_aI(int v[],int n)  
{  
    int i;  
    for (i=0;i<n;i++)  
        printf("%4d",v[i]);  
}
```

parametri di input

passaggio di array a function

notazione a puntatore

array come argomento (nella chiamata alla function):

✓ usare solo il nome dell'array

array come parametro (nell'intestazione di function):

✓ usare un puntatore

passaggio di array a function

Esempio

notazione a puntatore

```
...  
int a[10]={10,20,30,40,50,60,70,80,90,100};  
visualizza_aI(a, 10);  
...
```

```
void visualizza_aI(int *v, int n)  
{  
    int i;  
    for (i=0; i<n; i++)  
        printf("%4d", *(v+i));  
}
```


passaggio di array a function

Esempio

notazione a puntatore

```
...  
int a[10]={10,20,30,40,50,60,70,80,90,100};  
visualizza_aI(&a[0],10);  
...
```

```
void visualizza_aI(int *v,int n)  
{  
    int i;  
    for (i=0;i<n;i++)  
        printf("%4d",*(v+i));  
}
```

passaggio di array a function

Esempio

notazione mista

```
...  
int a[10]={10,20,30,40,50,60,70,80,90,100};  
visualizza_aI(a, 10);  
...
```

```
void visualizza_aI(int *v, int n)  
{  
    int i;  
    for (i=0;i<n;i++)  
        printf("%4d", v[i]);  
}
```

passaggio di array a function

la seguente chiamata è errata
(passa solo il valore di **a[0]**)

~~visualizza_a1(a[0],10);~~

se si passa il valore di un singolo elemento
dell'array, il passaggio è identico al passaggio di
una variabile scalare
(passaggio per valore)

```
int a[10]={10,20,30,40,50,60,70,80,90,100};  
int k;  
k = epari(a[0]);
```

```
int epari(int x);
```

esercizi

realizzare le seguenti function C

parametri di input

```
void visualizza_aD(double v[], int n)
```

✓ che visualizza sullo schermo un array di **double** di size **n**

parametro
di output

parametro
di input

```
void legge_da_tastiera_aD(double v[], int n)
```

✓ che legge da tastiera un array di **double** di size **n**

```
void visualizza_aD(double v[], int n)
{
    int i;
    for (i=0;i<n;i++)
        printf("\n%lf",v[i]);
    printf("\n");
}
```

```
void legge_da_tastiera_aD(double v[], int n)
{
    int i;
    printf("\n inserire %d valori (double)",n);
    for (i=0;i<n;i++)
    {
        printf("\n inserire %d-mo elemento: ",i);
        scanf("%lf",&v[i]);
    }
}
```