

- `int *p[4]` ?
- poiché `[]` ha priorità su `*`, `int *p[4]` si legge: `p` è un array di 4 elementi (ciascuno) `int *`, ovvero un array di 4 puntatori ad intero
- `int (*p)[4]` ?
- `int (*p)[4]` si legge: `p` è un puntatore ad un array di 4 interi

- `int p[4][2] ?`
- `p` è un array di 4 elementi, di cui ciascuno è un array di 2 interi
- il tipo di `p` è `int[4][2]`

# Array Multidimensionali



- Gli array possono avere più di una dimensione
  - Es. per rappresentare tabelle, matrici, oggetti multidimensionali
- tipo nome[indice1][indice2]...
- Es. `int x[3][4];`
- `int ar[2][3] = { {1, 2, 3}, {4, 5, 6} };`
- `int ar[2][3] = {1,2,3,4,5,6}; //identiche`  
// il [3] indica come “raggruppare per righe”

	Colonna 0	Colonna 1	Colonna 2	Colonna 3
Riga 0	a[ 0 ][ 0 ]	a[ 0 ][ 1 ]	a[ 0 ][ 2 ]	a[ 0 ][ 3 ]
Riga 1	a[ 1 ][ 0 ]	a[ 1 ][ 1 ]	a[ 1 ][ 2 ]	a[ 1 ][ 3 ]
Riga 2	a[ 2 ][ 0 ]	a[ 2 ][ 1 ]	a[ 2 ][ 2 ]	a[ 2 ][ 3 ]



Indice di colonna

Indice di riga

Nome dell'array

# Array Multidimensionali







- Inizializzazione:
- `int ar[2][3] = { {1, 2, 3}, {4, 5, 6} }; // = {1,2,3,4,5,6}`  
es. `a[1][0]`    `{0`  `1`  `}`



1020		
1021	1	<code>ar[0][0]</code>
1022	2	<code>ar[0][1]</code>
1023	3	<code>ar[0][2]</code>
1024	4	<code>ar[1][0]</code>
1025	5	<code>ar[1][1]</code>
1026	6	<code>ar[1][2]</code>
1027		




# Array Multidimensionali



- Inizializzazione:
- `int ar[2][3] = { {1, 2, 3}, {4, 5, 6} };`  
es. `a[1][0]`    `{0`  `1`  `}`

1020		
1021	1	<code>ar[0][0]</code> 
1022	2	<code>ar[0][1]</code>
1023	3	<code>ar[0][2]</code>
1024	4	<code>ar[1][0]</code> 
1025	5	<code>ar[1][1]</code>
1026	6	<code>ar[1][2]</code>
1027		






- Inizializzazione:
- `int ar[2][3] = { {1, 2, 3}, {4, 5, 6} };`  
 es. `a[1][0]`      `{0`  `1`  `}`  

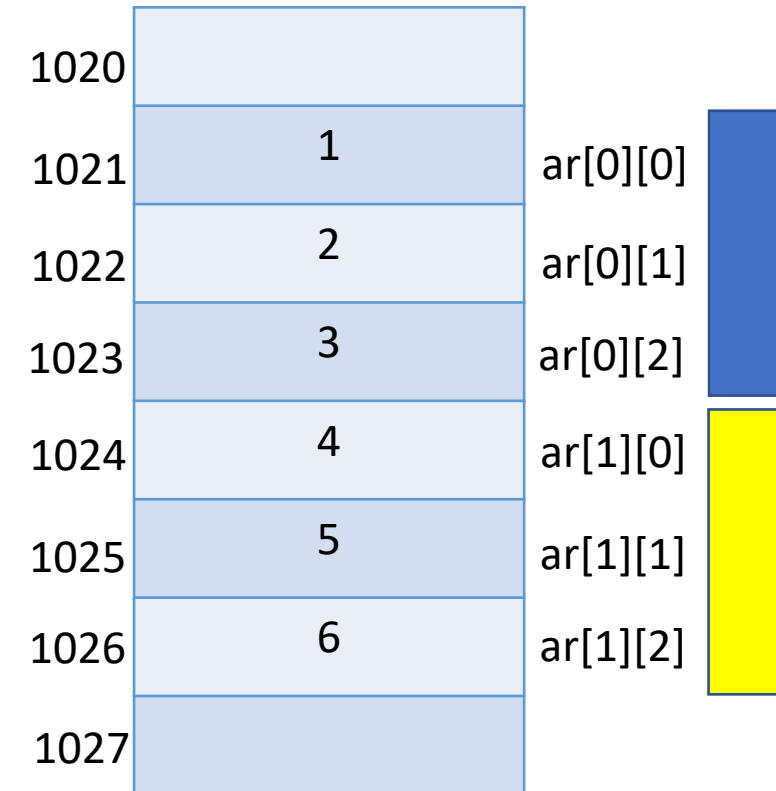
 `0`     `1`     `2`

1020		
1021	1	ar[0][0]
1022	2	ar[0][1]
1023	3	ar[0][2]
1024	4	ar[1][0]
1025	5	ar[1][1]
1026	6	ar[1][2]
1027		

# Array Multidimensionali



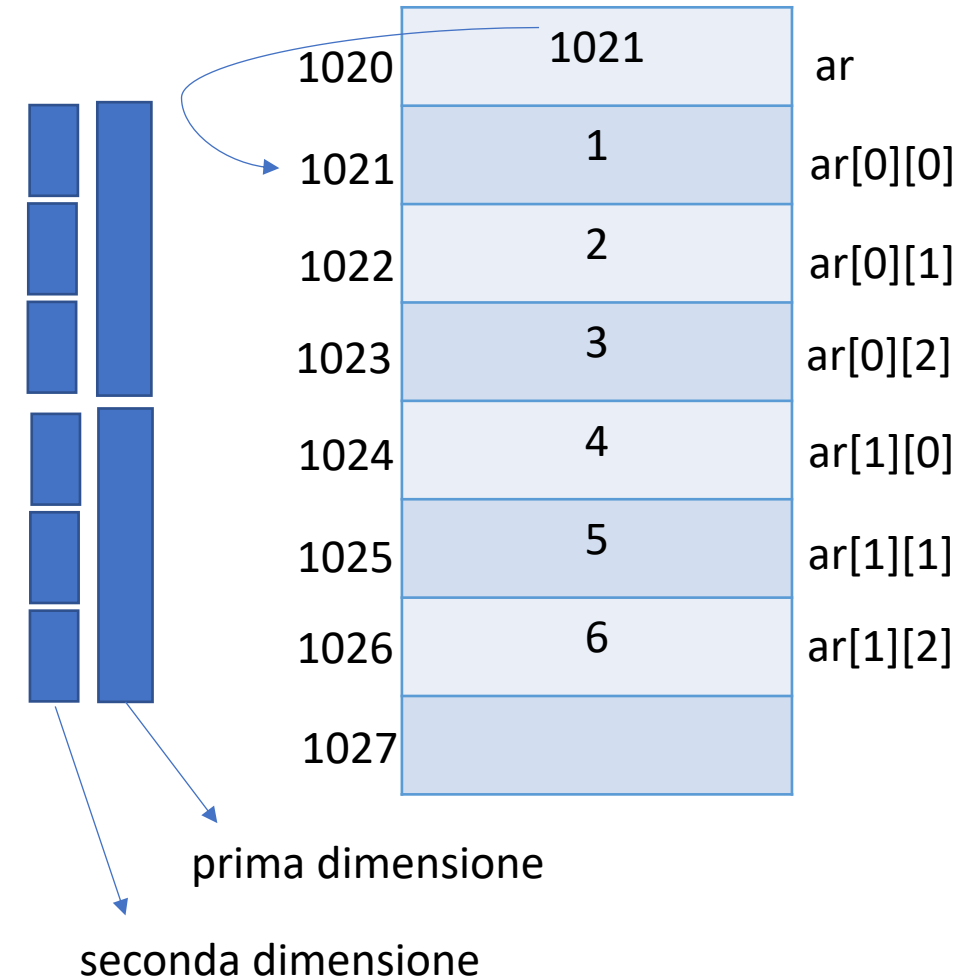
- Inizializzazione:
- `int ar[2][3] = { {1, 2, 3}, {4, 5, 6} };`  
es. `a[1][0]`    `{0`  `1`  `}`  
                  `0`  `1`  `2`  `}`
- Anche gli array multidimensionali sono salvati come una sequenza contigua di variabili, riga per riga. Le seguenti inizializzazioni sono equivalenti:
- `int ar[2][3] = {1, 2, 3, 4, 5, 6};`
- `int ar[][3] = {1, 2, 3, 4, 5, 6};` // solo la prima dimensione può essere lasciata in bianco



# Array Multidimensionali in Memoria



- `int ar[][3] = {1, 2, 3, 4, 5, 6};` // solo la prima dimensione può essere lasciata in bianco
- Perché le altre devono esserci? Se accedo a `ar[1][0]` devo sapere che devo saltare 3 elementi (`ar` è solo un puntatore al primo elemento dell'array)!
  - `int ar[][3]; ar[1][2] == 6` è l'elemento in posizione  $1*3 + 2$  in memoria (a partire da `ar`)
  - `int ar[][3]; int *b=(int*)ar; ar[1][2] == b[1*3 + 2]`
- Gli stessi ragionamenti si applicano ad array di più dimensioni: `int X[6][3][4];`
  - `X[1][2][3] →` elemento in posizione  $1*3*4 + 2*4 + 3$





# Matrici: Aritmetica dei Puntatori



- `int c[12];` definisce 12 variabili in memoria
- `c[i]` viene implementato dal C come `*(c+i*sizeof(int))`
- notate che `c` ha tipo `int[12]`, ma per eseguire `c+i`, `c` viene trasformato in `int*` (puntatore al tipo di ciascun elemento di `c`)
- In generale `c[i] → *(c+i*sizeof(TIPO DI CIASCUN ELEMENTO DI c))`
- se passiamo `c` come parametro ad una funzione, il C lo trasforma in un puntatore ad intero
  - `int f(int a[]);` `int f(int *a);` sono equivalenti; nel main si invoca con `f(c);`
  - in generale `c` viene trasformato ad un puntatore al tipo di ogni `c[i]`
- `int c[3][4];` definisce una matrice di 3 righe, ciascuna con 4 elementi
- sappiamo che `c[2]` equivale a `*(c+2*4);` per eseguire l'operazione il C deve sapere che la matrice `c` ha 4 colonne, questa informazione deve essere mantenuta nel tipo di `c`

c[ 0 ]	-45
c[ 1 ]	6
c[ 2 ]	0
c[ 3 ]	72
c[ 4 ]	1543
c[ 5 ]	-89
c[ 6 ]	0
c[ 7 ]	62
c[ 8 ]	-3
c[ 9 ]	1
c[ 10 ]	6453
c[ 11 ]	78

# Matrici: Aritmetica dei Puntatori



- In generale  $c[i] \rightarrow *(c+i*\text{sizeof}(\text{TIPO DI } c[i]))$
- `int c[3][4];` definisce una matrice di 3 righe, ciascuna con 4 elementi
- il tipo di `c` è `int[3][4]`, il tipo di `c[i]` è `int[4]`, un puntatore a `c` avrebbe tipo `int(*)[4]`, un puntatore a `c[i]` avrebbe tipo `int *`
- come possiamo trasformare `c[2][1]` utilizzando il fatto che  $c[i] = *(c+i*\text{sizeof}(\text{TIPO ELEMENTO DI } c))$ ?
- $c[2] \rightarrow *(c+2*\text{sizeof}(\text{int}[4])) = *(c+2*4*4) \rightarrow$  il tipo di `c`, per eseguire `c+32` diventa `int(*)[4]` (puntatore a `int[4]`)  $\rightarrow *(c+32)$  dereferenzia la variabile 32 byte dopo l'indirizzo di `c` ottenendo una variabile di tipo `int[4]`, `c[2]`

c[ 0 ]	-45
c[ 1 ]	6
c[ 2 ]	0
c[ 3 ]	72
c[ 4 ]	1543
c[ 5 ]	-89
c[ 6 ]	0
c[ 7 ]	62
c[ 8 ]	-3
c[ 9 ]	1
c[ 10 ]	6453
c[ 11 ]	78

# Matrici: Aritmetica dei Puntatori



- $c[2] \rightarrow *(c+2*\text{sizeof}(\text{int}[4])) = *(c+2*4*4) \rightarrow$  il tipo di  $c$ , per eseguire  $c+32$  diventa  $\text{int}(*)[4]$  (puntatore a  $\text{int}[4]$ )  $\rightarrow *(c+32)$  dereferenzia la variabile 32 byte dopo l'indirizzo di  $c$  ottenendo una variabile di tipo  $\text{int}[4]$ ,  $c[2]$

(cioè un array unidimensionale di tipo  $\text{int}[4]$ )

$\text{int} (*b)[4] = c+32; //$   $*b$  equivale a  $c[2]$

- $c[2][1] \rightarrow (c[2])[1] = *(c+2*\text{sizeof}(\text{int}[4]))[1] = (*(c+32))[1] = (*b)[1] = *(b+1*\text{sizeof}(\text{int})) = *(c+32+4) = 1$

$*b$  è di tipo  $\text{int}[4]$  quindi per sommare  $*b+1$  uso il puntatore a  $*b$ , ovvero  $b$

$*b$  è di tipo  $\text{int}[4]$  quindi  $(*b)[i]$  è di tipo  $\text{int}$

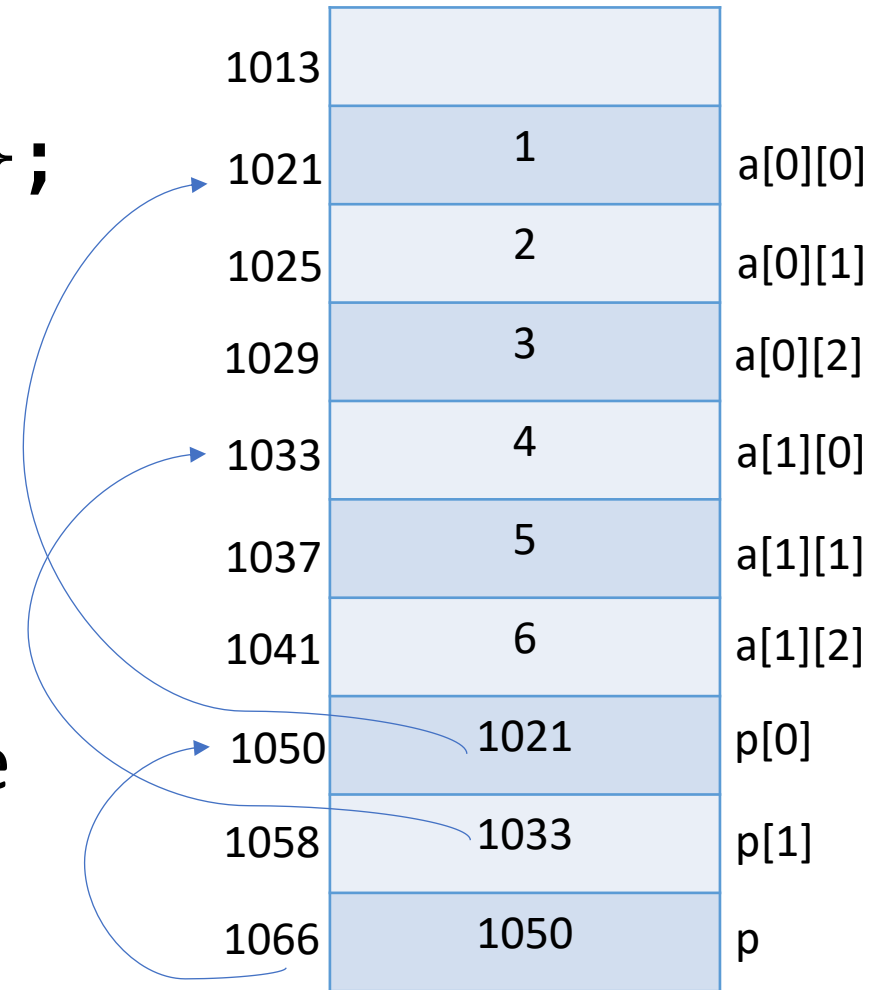
$c[0]$	-45
$c[1]$	6
$c[2]$	0
$c[3]$	72
$c[4]$	1543
$c[5]$	-89
$c[6]$	0
$c[7]$	62
$c[8]$	-3
$c[9]$	1
$c[10]$	6453
$c[11]$	78

# Matrici come Array di Puntatori



- `int a[][3] = { {1,2,3}, {4,5,6} };`
- `int *p[] = {a[0], a[1]};`

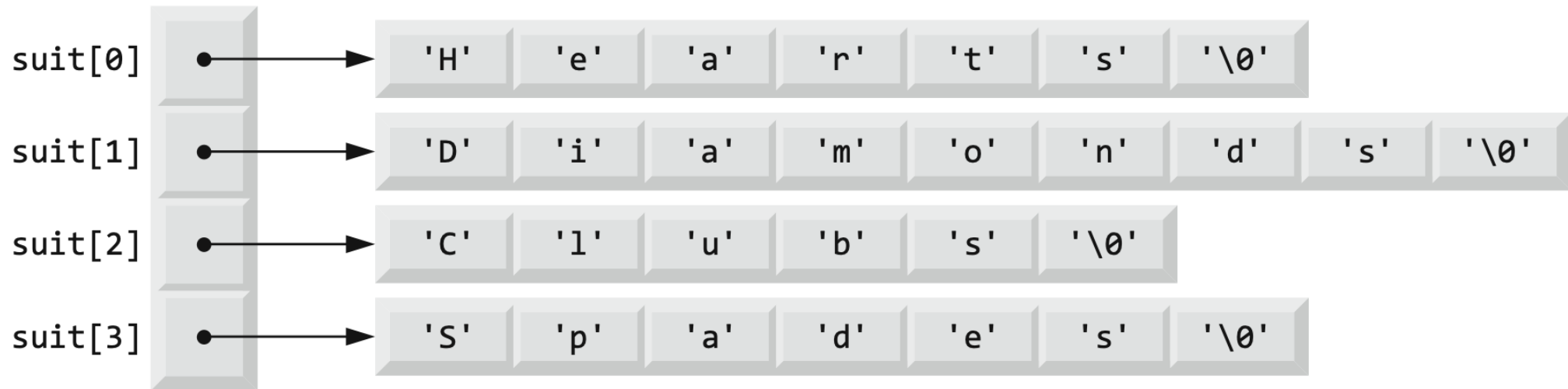
Tramite `p` ho rappresentato una matrice come un array di puntatori a vettori. Notate che ora potrei avere righe di dimensione diversa (ma dovrei ricordarmi la dimensione di ogni riga)



# Array Multidimensionali e Puntatori



- Gli array di puntatori sono più generali, permettono di avere righe di dimensione diversa:
- `char *suit[4] = {"Hearts", "Diamonds", "Clubs", "Spades"};`



- `int a[2][3] = { {1, 2, 3}, {4, 5, 6} };`
- `void stampa1(int a[][3], int righe);`
- `void stampa2(int** a, int righe, int colonne);`
- `void stampa3(int* a, int righe, int colonne)`

```
int *p[] = {a[0], a[1]};  
int *pp = a[0];
```

```
stampa1(a,2);  
stampa2(p,2,3);  
stampa3(pp,2,3);
```

- In pratica la cosa più complessa è azzeccare i tipi delle variabili, specialmente quando le passiamo come parametri ad una funzione.
- se abbiamo un puntatore ad un array unidimensionale, ad es. un `int*`, dobbiamo utilizzare `*(a+i*colonne+j)` per ottenere il valore dell'elemento `a(i,j)`
  - notate che non dovete usare `sizeof`, viene aggiunto automaticamente dal C
  - se l'array è tridimensionale, l'elemento `a(i,j,k)` è `*(a+i*righe*colonne + j*colonne + k)`
- se abbiamo una struttura a “più livelli”,
  - `int a[righe][colonne] = { {...}, {...}, ..., {...} }` oppure
  - `int*p[righe] = {a[0], a[1], ..., a[righe-1]}`
  - possiamo semplicemente usare la notazione `a[i][j]` (o `a[i][j][k]`), dove il primo indice, `i`, seleziona il vettore riga ed il secondo, `j`, l'elemento all'interno del vettore

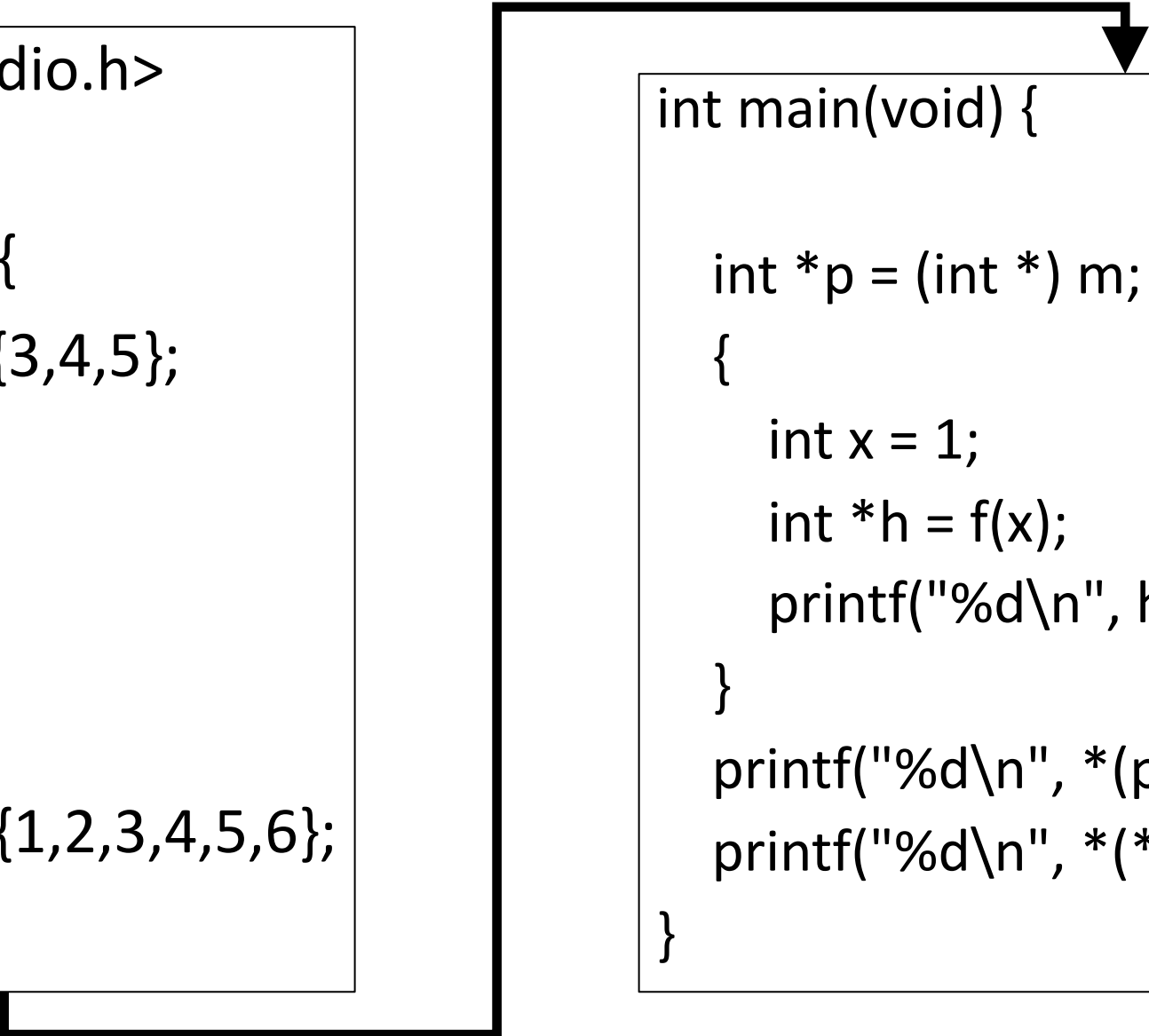
# Esercizio: cosa stampa?



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

```
int * f(int x) {  
    int a[3] = {3,4,5};  
    a[x]+=x;  
    return a;  
}
```

```
int m[3][2]={1,2,3,4,5,6};  
int x = 2;
```



```
int main(void) {  
  
    int *p = (int *) m;  
    {  
        int x = 1;  
        int *h = f(x);  
        printf("%d\n", h[1]);  
    }  
    printf("%d\n", *(p+x));  
    printf("%d\n", *(* (m+2)+1));  
}
```



- Date le seguenti matrici, trasformare l'operazione  $a[i][j]$  nell'operazione corrispondente che utilizza l'aritmetica dei puntatori:
  - `int *p = &a[0][0]; a[i][j]=*(p + ....)`
1. `int a[3][4]; a[2][1]`
  2. `float b[2][4]; b[2][3]`
  3. `int c[3][5][4]; c[2][1][3]`

- Date le seguenti matrici, trasformare l'operazione  $a[i][j]$  nell'operazione corrispondente che utilizza l'aritmetica dei puntatori:
  - `int *p = &a[0][0]; a[i][j]=*(p + ....)`
1. `int a[3][4]; int *p = &a[0][0]; a[2][1] → *(p+2*4+1)`
  2. `float b[2][4]; int *p = &b[0][0]; b[2][3] → *(p+2*4+3) → fuori dai limiti dell'array`
  3. `int c[3][5][4]; int *p = &c[0][0]; c[2][1][3] → *(p+2*5*4+1*4+3)`

- Date le seguenti matrici, trasformare l'operazione  $a[i][j]$  nell'operazione corrispondente che utilizza l'aritmetica dei puntatori:
  - `int *p = &a[0][0]; a[i][j]=*(p + ....)`
1. `int a[4][5]; p = &a[0][0]; a[2][1] → *(p+ ...)`
  2. `char a[3][2]; p = &a[0][0]; a[1][2] → *(p+ ...)`
  3. `char a[2][4][3]; p = &a[0][0]; a[1][2][2] → *(p+ ...)`

- Date le seguenti matrici, trasformare l'operazione  $a[i][j]$  nell'operazione corrispondente che utilizza l'aritmetica dei puntatori:
  - `int *p = &a[0][0]; a[i][j]=*(p + ....)`
1. `int a[4][5]; p = &a[0][0]; a[2][1] → *(p+2*5+1)`
  2. `char a[3][2]; p = &a[0][0]; a[1][2] → *(p+1*2+2) → l'array ha 2 colonne, non esiste l'elemento di colonna 2`
  3. `char a[2][4][3]; p = &a[0][0]; a[1][2][2] → *(p+1*4*3+2*3+2)`