

# Tecniche della Programmazione, lez.10

Recap su array monodimensionali

Gli array multidimensionali (multi == *bi* ...)

# Déjà vu - dieci per dieci

Programma che stampa i primi 100 numeri interi positivi, su 10 righe di 10 numeri ciascuna

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

```
int main () {  
    int i,j;
```

```
    for (INIT i; CHECK i; ADVANCE  
        USE i and j
```

```
    printf ("\nFINE programma\n");  
    return 0;  
}
```

```
0  1  2  3  4  5  6  7  8  9  
10 11 12 13 14 15 16 17 18 19  
20 21 22 23 24 25 26 27 28 29  
30 31 32 33 34 35 36 37 38 39  
40 41 42 43 44 45 46 47 48 49  
50 51 52 53 54 55 56 57 58 59  
60 61 62 63 64 65 66 67 68 69  
70 71 72 73 74 75 76 77 78 79  
80 81 82 83 84 85 86 87 88 89  
90 91 92 93 94 95 96 97 98 99  
FINE programma
```

In base a quanto detto a lezione, prova a scrivere e testare il programma.  
Comunque poi vedi Approfondimenti

Ah, dimenticavo. Vedi EserciziExtra in fondo alle slide, per un ulteriore esercizio. Prima provate a farlo. Poi trovate qualcosa nei complementi didattici, per confrontare le soluzioni e verificarle.

# Problema: Trasferimento Dimensionale

Nello Spazio esadimensionale\_intero ( $E_I$ ), i punti  $P^{EI}$  sono definiti da (6) coordinate intere.

Nello Spazio esadimensionale\_PARI ( $E_P$ ) i punti  $P^{EP}$  sono definiti da (6) coordinate intere che possono essere solo pari o nulle.

Due punti,  $P^{EP} \in E_P$  e  $P^{EI} \in E_I$ , si dicono **solidali**, sse per ogni  $i=1,\dots,6$  la coordinata  $i$ -esima di  $P^{EP}$  è

- uguale alla coordinata  $i$ -esima di  $P^{EI}$ , se questa è pari
- uguale a 0, se la coordinata  $i$ -esima di  $P^{EI}$  è dispari o nulla

ad esempio,  
questi due  
sono  
solidali

**punto<sup>EI</sup>** = (3259, 116, 5008, 5618, 47, 42)

**punto<sup>EP</sup>** = (0, 116, 5008, 5618, 0, 42)

Dato **punto<sup>EI</sup>**  $\in E_I$ , calcolare il punto solidale **punto<sup>EP</sup>**  $\in E_P$  (unico)

strutture dati?  
variabili e costanti?

Algoritmo?

# Problema del trasferimento di un punto<sup>EI</sup> in un punto<sup>EP</sup>

Rappresentiamo i due punti (vettori di coordinate) come array di interi;

punto<sup>EI</sup> = (3259, 116, 5008, 5618, 47, 42) (vettoreEI)

→

punto<sup>EP</sup> = (?, ?, ?, ?, ?, ?) (vettoreEP all'inizio)

punto<sup>EP</sup> = (0, 116, 5008, 5618, 0, 42) (vettoreEP alla fine)

```
#define N 6
int main () {
    int vettoreEI[N], vettoreEP[N];
```

Algoritmo (del programma)? ☺

Sottoproblemi?

Funzioni? ☺

# Problema del trasferimento di un punto<sup>EI</sup> in un punto<sup>EP</sup>

Rappresentiamo i due punti (vettori di coordinate) come array di interi;

punto<sup>EI</sup> = (3259, 116, 5008, 5618, 47, 42) (vettoreEI)

→

punto<sup>EP</sup> = (?, ?, ?, ?, ?, ?) (vettoreEP)

```
vettoreEP[0] = vettoreEI[0] ????  
vettoreEP[1] = vettoreEI[1] ??  
vettoreEP[2] = ??  
...  
vettoreEP[5] = ??
```

vettoreEP[i] = vettoreEI[i] oppure 0 ... **dipende**

punto<sup>EP</sup> = (0, 116, 5008, 5618, 0, 42) (vettoreEP)

```
#define N 6  
int main () {  
    int vettoreEI[N], vettoreEP[N];
```

**Algoritmo (del programma)?** ☺

Sottoproblemi?

Funzioni? ☺

# Problema del trasferimento di un punto<sup>EI</sup> in un punto<sup>EP</sup>

Rappresentiamo i due punti (vettori di coordinate) come array di interi;

punto<sup>EI</sup> = (3259, 116, 5008, 5618, 47, 42) (vettoreEI)

→

punto<sup>EP</sup> = (?, ?, ?, ?, ?, ?) (vettoreEP)

il nucleo del problema è nel **trasferimento parziale**, cioè nella copia degli elementi pari di vettoreEI in vettoreEP - nelle medesime posizioni, e inserimento di zero altrove in vettoreEP

cioè

assegnazione dell'elemento i-esimo di vettoreEP con

- l'elemento i-esimo di vettoreEI (se questo è pari)
- 0, se l'elemento i-esimo di vettoreEI è dispari o nullo

punto<sup>EP</sup> = (0, 116, 5008, 5618, 0, 42) (vettoreEP)

```
#define N 6
int main () {
    int vettoreEI[N], vettoreEP[N];
```

Algoritmo (del programma)? ☺

Sottoproblemi?

Funzioni? ☺

# Problema del trasferimento di un punto<sup>EI</sup> in un punto<sup>EP</sup>

```
caro/a utente, comunicami i dati del vettore:
```

```
- elemento [0]:3259  
- elemento [1]:116  
- elemento [2]:5008  
- elemento [3]:5618  
- elemento [4]:47  
- elemento [5]:42
```

```
ho letto il seguente vettore:
```

```
[3259, 116, 5008, 5618, 47, 42]
```

```
... hold on, transfer in progress ...
```

```
dopo il trasferimento dei pari, il secondo vettore e` come segue:
```

```
[0, 116, 5008, 5618, 0, 42]
```

```
FINE programma
```

**punto<sup>EP</sup>** = (0, 116, 5008, 5618, 0, 42) (vettoreEP alla fine)

```
#define N 6  
int main () {  
    int vettoreEI[N], vettoreEP[N];
```

Algoritmo (del programma)? ☺

Sottoproblemi?

Funzioni? ☺

# Problema del trasferimento di un punto<sup>EI</sup> in un punto<sup>EP</sup>

vettoreEI = (3259, 116, 5008, 5618, 47, 42)

→

vettoreEP = (0, 116, 5008, 5618, 0, 42)

```
#define N 6
```

```
int main () {
```

```
    int vettoreEI[N], vettoreEP[N];
```

## Algoritmo

0) ... strutture dati ...

1) lettura del primo vettore

2) stampa del primo vettore

3) trasferimento dal primo al secondo vettore

4) stampa del secondo vettore

5) FINE

Sottoproblemi?

Funzioni?





# Trasferimento tra array - main()

vettoreEI = (3259, 116, 5008, 5618, 47, 42)  
→ vettoreEP = (0, 116, 5008, 5618, 0, 42)

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

```
#define N 6
```

```
/* dichiarazioni delle funzioni usate dalla main() */
```

```
...
```

```
int main () {
```

```
    int vettoreEI[N], vettoreEP[N];
```

```
    printf ("caro/a utente, ... dati del vettore: \n");
```

```
    leggiVettore(vettoreEI);
```

```
    printf ("ho letto il seguente vettore:\n");
```

```
    stampaVettore(vettoreEI);
```

```
    printf ("\n... hold on, transfer in progress ...\n\n");
```

```
    trasferimentoPari(vettoreEI, vettoreEP);
```

```
    printf ("dopo il trasferimento ...: \n");
```

```
    stampaVettore(vettoreEP);
```

```
    printf ("\nFINE programma\n");
```

# Trasferimento tra array - main()

```
caro/a utente, comunicami i dati del vettore:
- elemento [0]:3259
- elemento [1]:116
- elemento [2]:5008
- elemento [3]:5618
- elemento [4]:47
- elemento [5]:42
ho letto il seguente vettore:
[3259, 116, 5008, 5618, 47, 42]
... hold on, transfer in progress ...
dopo il trasferimento dei pari, il secondo vettore e' come segue:
[0, 116, 5008, 5618, 0, 42]
FINE programma
```

```
printf ("caro/a utente, ... dati del vettore: \n");
leggiVettore(vettoreEI);
printf ("ho letto il seguente vettore:\n");
stampaVettore(vettoreEI);
printf ("\n... hold on, transfer in progress ... \n\n");
trasferimentoPari(vettoreEI, vettoreEP);
printf ("dopo il trasferimento ...: \n");
stampaVettore(vettoreEP);
printf ("\nFINE programma\n");
```

## Trasferimento tra array - algoritmo

vettoreEI = (3259, 116, 5008, 5618, 47, 42)



vettoreEP = (0, 116, 5008, 5618, 0, 42)

```
#define N 6
int main () {
    int vettoreEI[N], vettoreEP[N];
```

# Algoritmo

0) ...

## 1) lettura del primo vettore

## 2) stampa del primo vettore

### 3) trasferimento dal primo al secondo vettore

#### 4) stampa del secondo vettore

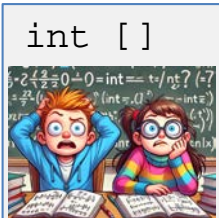
## 5) FINE

## Funzioni (sottoproblemi)

```
void leggiVettore(int []),
```

```
void stampaVettore(int[]),
```

```
void trasferisci( int [], int [])
```



# Trasferimento tra array - algoritmo

vettoreEI = (3259, 116, 5008, 5618, 47, 42)

→

vettoreEP = (0, 116, 5008, 5618, 0, 42)

```
#define N 6
int main () {
    int vettoreEI[N], vettoreEP[N];
```

## Algoritmo

- 0) ...
- 1) lettura del primo vettore
- 2) stampa del primo vettore
- 3) trasferimento dal primo al secondo vettore
- 4) stampa del secondo vettore
- 5) FINE

## Funzioni (sottoproblemi)

```
void leggiVettore(int []),
void stampaVettore(int[]),
void trasferisci( int [], int [])
```

int [] è il nome del tipo dell'array di interi, se omettiamo la dimensione va bene lo stesso, ma vedi slide successiva ...



# Programma per il trasferimento dimensionale: prototipi

```
vettoreEI = (3259, 116, 5008, 5618, 47, 42)
           →   vettoreEP = (0, 116, 5008, 5618, 0, 42)

#include <stdio.h>
#define N 6

/* dichiarazioni delle funzioni usate dalla main() */
void leggiVettore(int [N]);
void stampaVettore(int []);
void trasferimentoPari(int v1[N], int v2[N]);

int main () {
    int vettoreEI[N], vettoreEP[N];

    printf ("caro/a utente, ... dati del vettore: \n");
```

# Trasferimento tra array - prototipi

```
vettoreEI = (3259, 116, 5008, 5618, 47, 42)
          → vettoreEP = (0, 116, 5008, 5618, 0, 42)

#include <stdio.h>
#define N 6

/* dichiarazioni delle funzioni usate dalla main() */
void leggiVettore(int [N]);
void stampaVettore(int []);
void trasferimentoPari(int v1[N], int v2[N]);

int main () {
    int vettoreEI[N], vettoreEP[N];

    printf ("caro/a utente, ... dati del vettore: \n");
```

La dimensione si può omettere nella dichiarazione del parametro  
(ma metterla rende le cose più chiare a chi voglia riusare questa funzione, quindi è meglio)

## Trasferimento tra array - funzione di lettura

```
/* funzione di lettura di un vettore di N interi */  
void leggiVettore(int v[N]) {  
    int i;  
  
    }  
return;  
}
```



```
caro/a utente, comunicami i dati del vettore:  
- elemento [0]:3259  
- elemento [1]:116  
- elemento [2]:5008  
- elemento [3]:5618  
- elemento [4]:47  
- elemento [5]:42
```

# Trasferimento tra array - funzione di lettura

```
/* funzione di lettura di un vettore di N interi */  
void leggiVettore(int v[N]) {  
    int i;  
    for (i=0; i<N; i++) {  
        printf("- elemento [%d]:", i);  
        scanf("%d", &v[i]);  
    }  
    return;  
}
```



```
caro/a utente, comunicami i dati del vettore:  
- elemento [0]:3259  
- elemento [1]:116  
- elemento [2]:5008  
- elemento [3]:5618  
- elemento [4]:47  
- elemento [5]:42
```



# Trasferimento tra array - funzione di stampa

```
/* funzione di stampa di un vettore di N interi,  
nel formato [num, num, num ..., num] */
```

[3259, 116, 5008, 5618, 47, 42]

```
void stampaVettore(int v[N]) {  
    int i;  
    printf("[");  
    for (i=0; i<N; i++)
```



```
    printf("]\n");  
    return;  
}
```

```
- elemento [5]:42  
ho letto il seguente vettore:  
[3259, 116, 5008, 5618, 47, 42]
```

# Trasferimento tra array - funzione di stampa

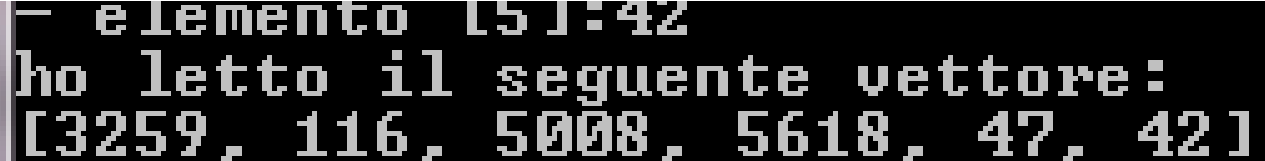
```
/* funzione di stampa di un vettore di N interi,  
nel formato [num, num, num ..., num] */
```

[3259, 116, 5008, 5618, 47, 42]

```
void stampaVettore(int v[N]) {  
    int i;  
    printf("[");  
    for (i=0; i<N; i++)
```

=> il primo viene stampato seguito da  
una ',' e da un '  
... l'ultimo no

```
    printf("]\n");  
    return;  
}
```



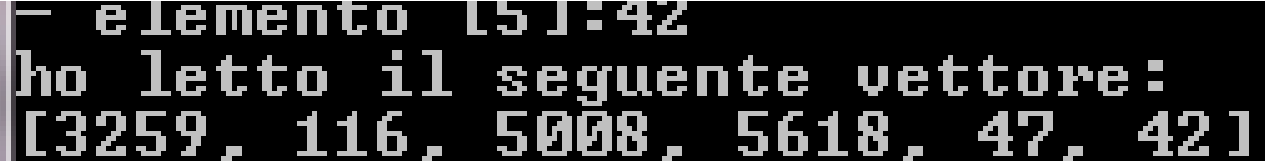
```
- elemento [5]:42  
ho letto il seguente vettore:  
[3259, 116, 5008, 5618, 47, 42]
```

# Trasferimento tra array - funzione di stampa

```
/* funzione di stampa di un vettore di N interi,  
nel formato [num, num, num ..., num] */
```

[3259, 116, 5008, 5618, 47, 42]

```
void stampaVettore(int v[N]) {  
    int i;  
    printf("[");  
    for (i=0; i<N; i++)  
        if (i==N-1)  
            printf("%d", v[i]);  
        else printf("%d, ", v[i]);  
  
    printf("]\n");  
    return;  
}
```

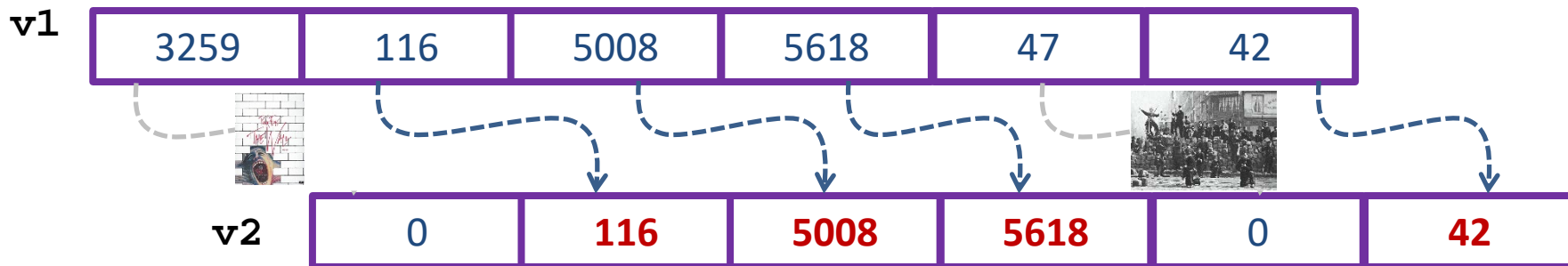


```
- elemento [5]:42  
ho letto il seguente vettore:  
[3259, 116, 5008, 5618, 47, 42]
```

# Trasferimento tra array - funzione finale

```
/* definizione della funzione che trasferisce i pari dal  
primo al secondo vettore e azzerava gli altri elementi del  
secondo vettore */
```

```
void trasferimentoPari(int v1[N], int v2[N]) {  
    int i;  
    for (i=0; i<N; i++)  
        if ((v1[i]%2)==0)  
            ☺  
        else ☺  
  
    return;  
}
```

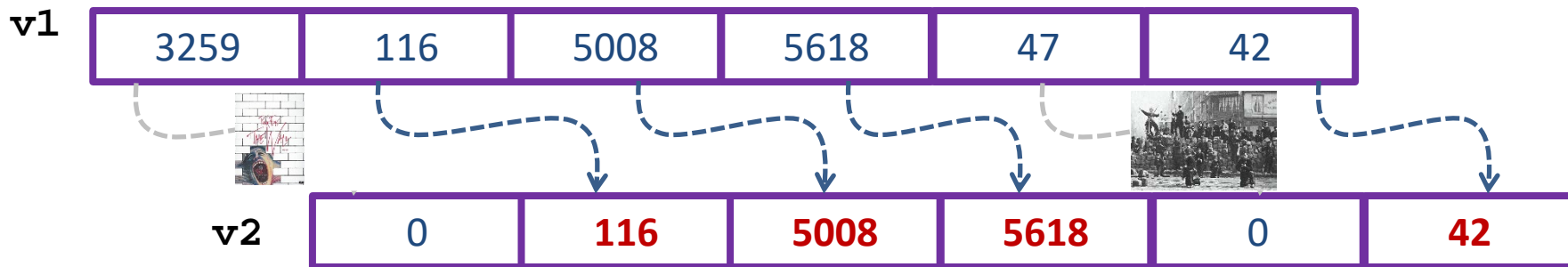


# Trasferimento tra array - funzione finale

```
/* definizione della funzione che trasferisce i pari dal  
primo al secondo vettore e azzerà gli altri elementi del  
secondo vettore */
```

vettoreEP[i] = vettoreEI[i] oppure 0 ... **dipende**

```
void trasferimentoPari(int v1[N], int v2[N]) {  
    int i;  
    for (i=0; i<N; i++)  
        if ((v1[i]%2)==0)  
            v2[i]=v1[i];  
        else  
            v2[i]=0;  
  
    return;  
}
```



# Array multidimensionali

Array → una riga → 1 indice

Array bidimensionale → righe per colonne → 2 indici

1° indice = di riga

2° indice = di colonna

`int mat[3][4]` dichiarazione di array bidimensionale di  
3 righe per 4 colonne ad elementi interi

<code>mat[0][0]</code>	<code>mat[0][1]</code>	<code>mat[0][2]</code>	<code>mat[0][3]</code>
<code>mat[1][0]</code>	<code>mat[1][1]</code>	<code>mat[1][2]</code>	<code>mat[1][3]</code>
<code>mat[2][0]</code>	<code>mat[2][1]</code>	<code>mat[2][2]</code>	<code>mat[2][3]</code>

`mat[i][j]` è l'elemento (la variabile semplice)  
corrispondente a riga `i` e colonna `j`

# Iterazioni su array bidimensionali

Le variabili di un array bidimensionale sono individuate da due indici, quindi bisogna far variare il primo indice e, per ogni suo valore, far variare il secondo indice - riferendosi così a tutti gli elementi.

Si tratta di due cicli annidati.

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

```
int mat[3][4];
```

```
...
```

```
for (i=0; i<3; i++)
```

```
    stampa riga i
```

i

0

j

0



```
0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
FINE programma
```

# Iterazioni su array bidimensionali - stampa (2 dimensioni)

Le variabili di un array bidimensionale sono individuate da due indici, quindi bisogna far variare il primo indice e, per ogni suo valore, far variare il secondo indice - riferendosi così a tutti gli elementi.

Si tratta di due **cicli annidati**.

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

fissato *i*,

qual è il codice che gestisce la *riga i*, cioè stampa gli elementi  
`mat[i][0]`, `mat[i][1]`, `mat[i][2]`, `mat[i][3]`

...  
`for (i=0; i<3; i++)`

...

```
printf("mat[%d][%d]: %d\n",  
      indici ..., mat[i][j]);
```



# Algoritmo per la stampa di una matrice (tsk array bidimensionale)

```
#define N 3  
#define M 4  
int mat[N][M];
```

e adesso una stampa migliore ...

```
mat[0][0]:  1  mat[0][1]:  2  mat[0][2]:  3  mat[0][3]:  4  
mat[1][0]: 10  mat[1][1]: 20  mat[1][2]: 30  mat[1][3]: 40  
mat[2][0]: 100 mat[2][1]: 200 mat[2][2]: 300 mat[2][3]: 400
```

- 0) ... N, M, ...
- 1) ripetere, per tutte le righe
  - 1.1) stampa la riga i-esima
- 2) fine stampa

# Algoritmo per la stampa di una matrice: raffinamento

```
#define N 3
#define M 4
int mat[N][M];
```

e adesso una stampa migliore ...

```
mat[0][0]:  1  mat[0][1]:  2  mat[0][2]:  3  mat[0][3]:  4
mat[1][0]: 10  mat[1][1]: 20  mat[1][2]: 30  mat[1][3]: 40
mat[2][0]: 100 mat[2][1]: 200 mat[2][2]: 300 mat[2][3]: 400
```

Raffinamento ...

- 0) ... N, M, indice\_riga, indice\_colonna, mat[][] ...
- 1) ripetere, con indice\_riga=0...N-1
  - 1.1) stampa mat[i][0] ... mat[i][M-1]
- 2) fine stampa

# Torniamo al codice

Le variabili di un array bidimensionale sono individuate da due indici, quindi bisogna far variare il primo indice e, per ogni suo valore, far variare il secondo indice - riferendosi così a tutti gli elementi.

Si tratta di due **cicli annidati**.

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

fissato *i*,  
questo è il codice che gestisce la *riga i*, cioè gli elementi  
*mat[i][0], mat[i][1], mat[i][2], mat[i][3]*

*j*                      *j*                      *j*                      *j*

...

```
for (i=0; i<3; i++)  
    for (j=0; j<4; j++)  
        printf("mat[%d][%d]: %d\n",  
               i, j, mat[i][j]);
```

# Iterazioni su array bidimensionali

Le variabili di un array bidimensionale sono individuate da due indici, quindi bisogna far variare il primo indice e, per ogni suo valore, far variare il secondo indice - riferendosi così a tutti gli elementi.

Si tratta di due **cicli annidati**.

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

```
int mat[3][4];
```

```
...
```

```
for (i=0; i<3; i++)
```

```
    for (j=0; j<4; j++)
```

```
        printf("mat[%d][%d]: %d\n",
```

```
            i, j, mat[i][j]);
```

i

0

j

0

Eseguire questo codice in modo simulato, passo passo ... tenendo anche traccia dell'output via via prodotto.  
E poi vedi Esercizi

# Algoritmo per la stampa di una matrice: altro raffinamento

```
#define N 3
#define M 4
int mat[N][M];
```

e adesso una stampa migliore ...

```
mat[0][0]: 1  mat[0][1]: 2  mat[0][2]: 3  mat[0][3]: 4
mat[1][0]: 10 mat[1][1]: 20 mat[1][2]: 30 mat[1][3]: 40
mat[2][0]: 100 mat[2][1]: 200 mat[2][2]: 300 mat[2][3]: 400
```

## Raffinamento ulteriore...

- 0) ... N, M, indice\_riga, indice\_colonna, mat[][] ...
- 1) ripetere, con indice\_riga=0...N-1
  - 1.1) ripetere, con indice\_colonna=0, ..., M-1
    - 1.1.1) stampa mat[indice\_riga][indice\_colonna]
- 2) fine stampa

# Algoritmo per la stampa di una matrice: ulteriore raffinamento

```
#define N 3
#define M 4
int mat[N][M];
```

e adesso una stampa migliore ...

```
mat[0][0]: 1  mat[0][1]: 2  mat[0][2]: 3  mat[0][3]: 4
mat[1][0]: 10 mat[1][1]: 20 mat[1][2]: 30 mat[1][3]: 40
mat[2][0]: 100 mat[2][1]: 200 mat[2][2]: 300 mat[2][3]: 400
```

Questione pratica, e ultimo raffinamento: dopo ogni riga bisogna andare a capo, e poi i valori vanno stampati incolonnati ...

0) ... N, M, indice\_riga, indice\_colonna, mat[][] ...

1) ripetere, con indice\_riga=0...N-1

1.1) ripetere, con indice\_colonna=0, ..., M-1

1.1.1) stampa mat[indice\_riga][indice\_colonna], nella forma

"mat[indice\_riga][indice\_colonna]: il suo valore, su tre colonne

1.2) stampare '\n'

2) fine stampa

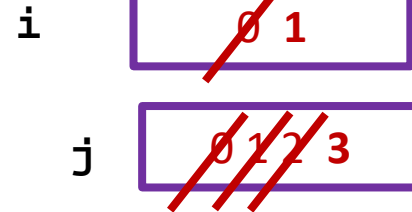
# Dall'algoritmo al codice per la stampa di una matrice

## Una stampa migliore

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	<b>mat[1][3]</b>
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

A) Per ogni valore di i (per ogni riga)

```
for (i=0; i<3; i++) {  
    for (j=0; j<4; j++)  
        printf(" mat[%d][%d]: %3d ",  
               i, j, mat[i][j]);  
    putchar(' \n');
```



A.1) Stampa gli elementi mat[i][j]  
Per j = 0...3

A.2) E poi va a capo

e adesso una stampa migliore ...

```
mat[0][0]: 1  mat[0][1]: 2  mat[0][2]: 3  mat[0][3]: 4  
mat[1][0]: 10 mat[1][1]: 20 mat[1][2]: 30 mat[1][3]: 40  
mat[2][0]: 100 mat[2][1]: 200 mat[2][2]: 300 mat[2][3]: 400
```

# Array bidimensionali: uso di costanti per le dimensioni

```
#define N 3  
#define M 4
```

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	<b>mat[1][1]</b>	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

```
for (i=0; i<N; i++) {  
    for (j=0; j<M; j++)  
        printf(" mat[%d][%d]: %3d ",  
               i, j, mat[i][j]);  
    putchar( '\n' );  
}
```

i

1

j

1

**NB ... Nelle prossime slide usiamo 3 e 4 in esempi ad hoc ...  
MA dobbiamo abituarci ad usare le costanti per le dimensioni!**

Ok, tempo di esercizi (per i quali potreste trovare qualcosa nella directory dei **complementi didattici**).

- 1) Scrivere un programma che riceve in input una matrice NxM e la stampa per bene come visto sopra.
- 2) Idem, ma realizzando la stampa mediante un ciclo while. Conviene riscrivere l'algoritmo ricordando bene le relazioni tra while e for ... su questo, una slide e` nella zona **Esercizi**



# Inizializzazione di un array bidimensionale

NB la lettura da input di un array bidimensionale scandisce gli elementi così come visto per la stampa ... solo che si usa `scanf()` ...

L'inizializzazione in definizione è possibile, così come per gli array monodimensionali:

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

MA si può omettere solo la prima dimensione. Per il resto, ci sono varie possibilità ...

```
int mat[3][4] = { {1, 2, 3, 4},  
                  {10, 20, 30, 40},  
                  {100, 200, 300, 400} };
```

```
int matt[][4] = { {1, 2, 3}, {10, 20, 30},  
                  {100, 200, 300, 400} };
```

```
int mattt[3][4] = { 1, 2, 3, 4, 10, 20, 30,  
                    40, 100, 200, 300, 400 };
```

# Inizializzazione di un array bidimensionale

```
mat[0][0]: 1  mat[0][1]: 2  mat[0][2]: 3  mat[0][3]: 4
mat[1][0]: 10 mat[1][1]: 20 mat[1][2]: 30 mat[1][3]: 40
mat[2][0]: 100 mat[2][1]: 200 mat[2][2]: 300 mat[2][3]: 400
```

```
matt[0][0]: 1  matt[0][1]: 2  matt[0][2]: 3  matt[0][3]: 0
matt[1][0]: 10 matt[1][1]: 20 matt[1][2]: 30 matt[1][3]: 0
matt[2][0]: 100 matt[2][1]: 200 matt[2][2]: 300 matt[2][3]: 400
```

```
mattt[0][0]: 1  mattt[0][1]: 2  mattt[0][2]: 3  mattt[0][3]: 4
mattt[1][0]: 10 mattt[1][1]: 20 mattt[1][2]: 30 mattt[1][3]: 40
mattt[2][0]: 100 mattt[2][1]: 200 mattt[2][2]: 300 mattt[2][3]: 400
```

```
int mat[3][4] = { {1, 2, 3, 4},
                  {10, 20, 30, 40},
                  {100, 200, 300, 400} };
```

```
int matt[][4] = { {1, 2, 3}, {10, 20, 30},
                  {100, 200, 300, 400} };
```

```
int mattt[3][4] = { 1, 2, 3, 4, 10, 20, 30,
                    40, 100, 200, 300, 400 };
```

sperimentare queste inizializzazioni in un nuovo programma, ottenuto da quello precedente. L'utilità sta nel riflettere su come gli elementi dell'array bidimensionale sono memorizzati nella RAM ... [Poi Vedi Approfondimenti](#)

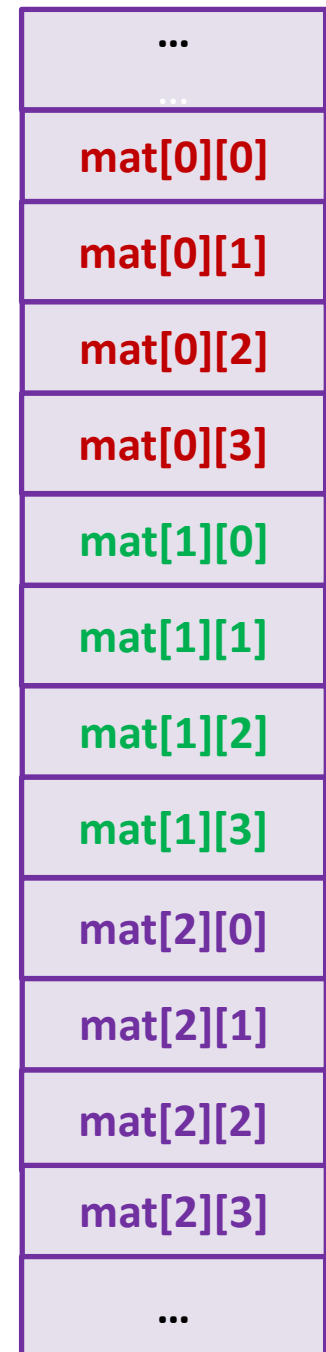
# Si', ma in memoria?

La disposizione in **memoria** è sequenziale, riga per riga

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

```
int mat[3][4] = { {1, 2, 3, 4},  
                  {10, 20, 30, 40},  
                  {100, 200, 300, 400} };
```

```
for (i=0; i<3; i++)  
    for (j=0; j<4; j++)  
        printf("l'elemento mat[%d][%d]  
è di %d byte, ha indirizzo %p  
e valore %d\n",  
i, j, sizeof(mat[i][j]),  
&mat[i][j], mat[i][j]);
```



# Tecniche della Programmazione, lez. 10

- Approfondimenti

# Déjà vu - dieci per dieci

Programma che stampa i primi 100 numeri interi positivi, su 10 righe di 10 numeri ciascuna

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

```
int main () {  
    int i,j;
```

```
    for (INIT i; CHECK i; ADVANCE  
        USE i and j
```

```
    printf ("\nFINE programma\n");  
    return 0;  
}
```

```
0  1  2  3  4  5  6  7  8  9  
10 11 12 13 14 15 16 17 18 19  
20 21 22 23 24 25 26 27 28 29  
30 31 32 33 34 35 36 37 38 39  
40 41 42 43 44 45 46 47 48 49  
50 51 52 53 54 55 56 57 58 59  
60 61 62 63 64 65 66 67 68 69  
70 71 72 73 74 75 76 77 78 79  
80 81 82 83 84 85 86 87 88 89  
90 91 92 93 94 95 96 97 98 99  
FINE programma
```

# Esercizio - dieci per dieci - annidamento di cicli

Programma che stampa i primi 100 numeri interi positivi, su 10 righe di 10 numeri ciascuna

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

```
int main () {  
    int i,j;
```

```
0  1  2  3  4  5  6  7  8  9  
10 11 12 13 14 15 16 17 18 19  
20 21 22 23 24 25 26 27 28 29  
30 31 32 33 34 35 36 37 38 39  
40 41 42 43 44 45 46 47 48 49  
50 51 52 53 54 55 56 57 58 59  
60 61 62 63 64 65 66 67 68 69  
70 71 72 73 74 75 76 77 78 79  
80 81 82 83 84 85 86 87 88 89  
90 91 92 93 94 95 96 97 98 99  
  
FINE programma
```

## ALGORITMO

- ciclo con i che va da 0 a 9
  - ad ogni iterazione ( $i=0, i=1, \dots, i=9$ )
    - viene stampata una riga di numeri che sono quelli che vanno da  $i*10$  a  $i*10 + 9$
    - fine riga (insomma: si va a capo)

```
printf ("\nFINE programma\n");  
return 0;  
}
```

- ☺ prova a spiegare cosa avviene ad ogni iterazione del ciclo su i
- ☺ prova a spiegare cosa avviene ad ogni iterazione del ciclo su j
  - ☺ (cioè, cosa succede per  $j=0$ ? Cosa per  $j=1$ ? 2 3 ... 9?

# Esercizio - dieci per dieci - annidamento

Programma che stampa i primi 100 numeri interi positivi, su 10 righe di 10 numeri ciascuna

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

```
int main () {  
    int i,j;
```

```
0  1  2  3  4  5  6  7  8  9  
10 11 12 13 14 15 16 17 18 19  
20 21 22 23 24 25 26 27 28 29  
30 31 32 33 34 35 36 37 38 39  
40 41 42 43 44 45 46 47 48 49  
50 51 52 53 54 55 56 57 58 59  
60 61 62 63 64 65 66 67 68 69  
70 71 72 73 74 75 76 77 78 79  
80 81 82 83 84 85 86 87 88 89  
90 91 92 93 94 95 96 97 98 99
```

FINE programma

## ALGORITMO

### 1) ciclo con i che va da 0 a 9

- ad ogni iterazione viene stampata una riga di numeri che sono quelli che vanno da  $i*10$  a  $i*10 + 9$

CIOE`

#### 1.1) ciclo con j che va da 0 a 9

- ad ogni iterazione viene stampato il numero  $i*10 + j$

#### 1.2) andare a capo per la prossima riga di numeri

- fine ...

```
printf ("\nFINE programma\n");  
return 0;  
}
```

# Esercizio - dieci per dieci

Programma che stampa i primi 100 numeri interi positivi, su 10 righe di 10 numeri ciascuna

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

```
int main () {  
    int i,j;
```

```
    for (i=0; i<10; i++) {
```

```
        /* stampa la riga i-esima di 10 numeri */
```

```
        for (j=0; j<10; j++) /* 1) */
```

```
            printf ("%2d ", i*10 + j);
```

```
        printf ("\n"); /* 2) fine riga */
```

```
    }
```

```
    printf ("\nFINE programma\n");
```

```
    return 0;
```

```
}
```

```
0  1  2  3  4  5  6  7  8  9  
10 11 12 13 14 15 16 17 18 19  
20 21 22 23 24 25 26 27 28 29  
30 31 32 33 34 35 36 37 38 39  
40 41 42 43 44 45 46 47 48 49  
50 51 52 53 54 55 56 57 58 59  
60 61 62 63 64 65 66 67 68 69  
70 71 72 73 74 75 76 77 78 79  
80 81 82 83 84 85 86 87 88 89  
90 91 92 93 94 95 96 97 98 99
```

```
FINE programma
```



# Si', ma in memoria?

La disposizione in memoria è sequenziale, riga per riga

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

```
int mat[3][4] = { {1, 2, 3, 4},  
                  {10, 20, 30, 40},  
                  {100, 200, 300, 400} };
```

```
for (i=0; i<3; i++)  
    for (j=0; j<4; j++)  
        printf("l'elemento mat[%d][%d]  
        è di %d byte, ha indirizzo %p  
        e valore %d\n",  
        i, j, sizeof(mat[i][j]),  
        &mat[i][j], mat[i][j]);
```

...
mat[0][0]
mat[0][1]
mat[0][2]
mat[0][3]
mat[1][0]
mat[1][1]
mat[1][2]
mat[1][3]
mat[2][0]
mat[2][1]
mat[2][2]
mat[2][3]
...

# Si`, ma in memoria?

La disposizione in memoria è sequenziale, riga per riga

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

```
int mat[3][4] = { {1, 2, 3, 4},  
{10, 20, 30, 40},  
{100, 200, 300, 400} };
```

```
mat[0][0] 4 byte, ha indirizzo 0028FF00 e valore 1  
mat[0][1] 4 byte, ha indirizzo 0028FF04 e valore 2  
mat[0][2] 4 byte, ha indirizzo 0028FF08 e valore 3  
mat[0][3] 4 byte, ha indirizzo 0028FF0C e valore 4  
mat[1][0] 4 byte, ha indirizzo 0028FF10 e valore 10  
mat[1][1] 4 byte, ha indirizzo 0028FF14 e valore 20  
mat[1][2] 4 byte, ha indirizzo 0028FF18 e valore 30  
mat[1][3] 4 byte, ha indirizzo 0028FF1C e valore 40  
mat[2][0] 4 byte, ha indirizzo 0028FF20 e valore 100  
mat[2][1] 4 byte, ha indirizzo 0028FF24 e valore 200  
mat[2][2] 4 byte, ha indirizzo 0028FF28 e valore 300  
mat[2][3] 4 byte, ha indirizzo 0028FF2C e valore 400
```

...
mat[0][0] = 1
mat[0][1]
mat[0][2] = 3
mat[0][3]
mat[1][0] = 10
mat[1][1] = 20
mat[1][2]
mat[1][3]
mat[2][0]
mat[2][1] = 200
mat[2][2]
mat[2][3] = 400
...

# Tecniche della Programmazione, lez. 10

- Esercizi

# Iterazioni su array bidimensionali

Le variabili di un array bidimensionale sono individuate da due indici, quindi bisogna far variare il primo indice e, per ogni suo valore, far variare il secondo indice - riferendosi così a tutti gli elementi.

Si tratta di due cicli annidati.

<b>mat[0][0]</b>	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

```
int mat[3][4];
```

```
...
```

```
for (i=0; i<3; i++)
```

```
    for (j=0; j<4; j++)
```

```
        printf("mat[%d][%d]: %d\n",
```

```
                i, j, mat[i][j]);
```

i

0

j

0

# Iterazioni su array bidimensionali

Le variabili di un array bidimensionale sono individuate da due indici, quindi bisogna far variare il primo indice e, per ogni suo valore, far variare il secondo indice - riferendosi così a tutti gli elementi.

Si tratta di due cicli annidati.

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

```
int mat[3][4];
```

```
...
```

```
for (i=0; i<3; i++)
```

```
    for (j=0; j<4; j++)
```

```
        printf("mat[%d][%d]: %d\n",
```

```
            i, j, mat[i][j]);
```

i

0

j

~~0~~ 1

# Iterazioni su array bidimensionali

Le variabili di un array bidimensionale sono individuate da due indici, quindi bisogna far variare il primo indice e, per ogni suo valore, far variare il secondo indice - riferendosi così a tutti gli elementi.

Si tratta di due cicli annidati.

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

```
int mat[3][4];
```

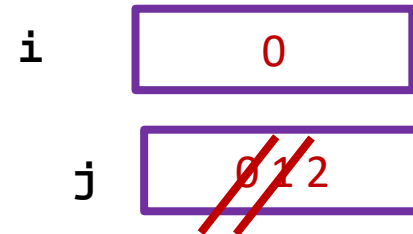
```
...
```

```
for (i=0; i<3; i++)
```

```
    for (j=0; j<4; j++)
```

```
        printf("mat[%d][%d]: %d\n",
```

```
                i, j, mat[i][j]);
```



# Iterazioni su array bidimensionali

Le variabili di un array bidimensionale sono individuate da due indici, quindi bisogna far variare il primo indice e, per ogni suo valore, far variare il secondo indice - riferendosi così a tutti gli elementi.

Si tratta di due cicli annidati.

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

```
int mat[3][4];
```

```
...
```

```
for (i=0; i<3; i++)
```

```
    for (j=0; j<4; j++)
```

```
        printf("mat[%d][%d]: %d\n",
```

```
            i, j, mat[i][j]);
```

i

0

j

~~0~~ ~~1~~ ~~2~~ 3

# Iterazioni su array bidimensionali

Le variabili di un array bidimensionale sono individuate da due indici, quindi bisogna far variare il primo indice e, per ogni suo valore, far variare il secondo indice - riferendosi così a tutti gli elementi.

Si tratta di due cicli annidati.

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

```
int mat[3][4];
```

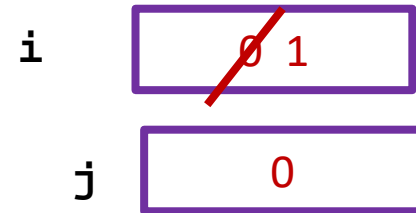
```
...
```

```
for (i=0; i<3; i++)
```

```
    for (j=0; j<4; j++)
```

```
        printf("mat[%d][%d]: %d\n",
```

```
                i, j, mat[i][j]);
```





# Iterazioni su array bidimensionali

Le variabili di un array bidimensionale sono individuate da due indici, quindi bisogna far variare il primo indice e, per ogni suo valore, far variare il secondo indice - riferendosi così a tutti gli elementi.

Si tratta di due cicli annidati.

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

```
int mat[3][4];
```

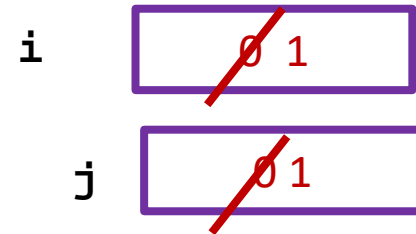
```
...
```

```
for (i=0; i<3; i++)
```

```
    for (j=0; j<4; j++)
```

```
        printf("mat[%d][%d]: %d\n",
```

```
                i, j, mat[i][j]);
```



# Iterazioni su array bidimensionali

Le variabili di un array bidimensionale sono individuate da due indici, quindi bisogna far variare il primo indice e, per ogni suo valore, far variare il secondo indice - riferendosi così a tutti gli elementi.

Si tratta di due cicli annidati.

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

```
int mat[3][4];
```

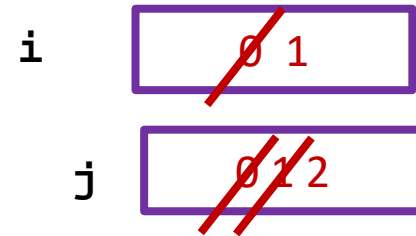
```
...
```

```
for (i=0; i<3; i++)
```

```
    for (j=0; j<4; j++)
```

```
        printf("mat[%d][%d]: %d\n",
```

```
                i, j, mat[i][j]);
```



# Iterazioni su array bidimensionali

Le variabili di un array bidimensionale sono individuate da due indici, quindi bisogna far variare il primo indice e, per ogni suo valore, far variare il secondo indice - riferendosi così a tutti gli elementi.

Si tratta di due cicli annidati.

mat[0][0]	mat[0][1]	mat[0][2]	mat[0][3]
mat[1][0]	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	mat[2][1]	mat[2][2]	mat[2][3]

```
int mat[3][4];
```

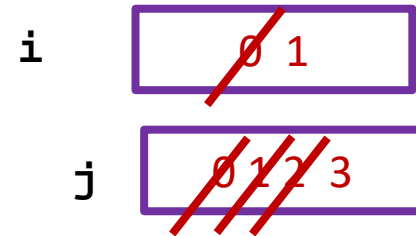
```
...
```

```
for (i=0; i<3; i++)
```

```
    for (j=0; j<4; j++)
```

```
        printf("mat[%d][%d]: %d\n",
```

```
                i, j, mat[i][j]);
```



# Iterazioni su array bidimensionali

```
elemento mat[0][0]: 1
elemento mat[0][1]: 2
elemento mat[0][2]: 3
elemento mat[0][3]: 4
elemento mat[1][0]: 10
elemento mat[1][1]: 20
elemento mat[1][2]: 30
elemento mat[1][3]: 40
elemento mat[2][0]: 100
elemento mat[2][1]: 200
elemento mat[2][2]: 300
elemento mat[2][3]: 400
```

bidimensionale sono individuate da due variare il primo indice e, per ogni suo secondo indice - riferendosi così a tutti

1	2	3	4
10	mat[1][1]	mat[1][2]	mat[1][3]
mat[2][0]	200	mat[2][2]	400

```
int mat[3][4];
```

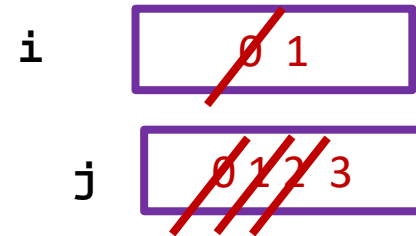
```
...
```

```
for (i=0; i<3; i++)
```

```
    for (j=0; j<4; j++)
```

```
        printf("mat[%d][%d]: %d\n",
```

```
                i, j, mat[i][j]);
```



# Algoritmo per la stampa di una matrice

"while like"

```
#define N 3
#define M 4
int mat[N][M];
```

Algoritmo (per la parte di codice che stampa la matrice)

```
0) ... N, M, mat, indice_riga, indice_colonna
1) indice_riga = 0;
2) mentre indice_riga < N
    2.1) indice_colonna = 0
    2.2) mentre indice_colonna < M
        2.2.1) stampa mat[indice_riga][indice_colonna]\n
        2.2.2) indice_colonna += 1
    2.3) indice_riga += 1
3) fine stampa
```

# Tecniche della Programmazione, lez. 10

- Esercizio Extra

# esercizio extra - su array

"in previsione della EG ..."

Scrivere un programma che mostri l'output in immagine.  
Inizializzare l'array in definizione.

```
caro/a utente, l'array e'  
( 32.59  116  50.08  56.18  47  42 )  
  
... sto calcolando il massimo  
... ci sono quasi  
... ecco ...  
ok, il massimo e` 116  
FINE
```