Tecniche della Programmazione, lez.10

Un esercizio con gli array

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

Punto su algoritmi

Problema

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,…,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

$punto^{EI}$ = (3259, 116, 5008, 5618, 47, 42)

$punto^{EP}$ = (0, 116, 5008, 5618, 0, 42)

Dato $punto^{EI} \in E\_I$, calcolare il punto solidale $punto^{EP} \in E\_P$ (unico)

strutture dati?
variabili e costanti?

Algoritmo?

Problema della trasformazione di un punto^{EI} in un punto^{EP}

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

punto^{EI} = (3259, 116, 5008, 5618, 47, 42) (vettore1)

→

punto^{EP} = (?, ?, ?, ?, ?, ?) (vettore2 all'inizio)

punto^{EP} = (0, 116, 5008, 5618, 0, 42) (vettore2 alla fine)

```
#define N 6
int main () {
  int vettore1[N], vettore2[N];
```

Algoritmo (del programma)? ☺
Sottoproblemi?
Funzioni? ☺

Problema della trasformazione di un puntoEI in un puntoEP

vettore1 = (3259, 116, 5008, 5618, 47, 42)
→ vettore2 = (0, 116, 5008, 5618, 0, 42)

```
#define N 6
int main () {
  int vettore1[N], vettore2[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? ☺

Problema della trasformazione di un puntoEI in un puntoEP

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

puntoEI = (**3259, 116, 5008, 5618, 47, 42**) (vettore1)

→

puntoEP = (**?, ?, ?, ?, ?, ?**) (vettore2)

il nucleo del problema è nel **trasferimento parziale, cioè nella** copia degli elementi pari di vettore1 in vettore2 – nelle medesime posizioni, e inserimento di zero altrove in vettore2

cioè

assegnazione dell'elemento i-esimo di vettore2 con
- l'elemento i-esimo di vettore1 (se questo è pari)
- 0, se l'elemento i-esimo di vettore1 è dispari o nullo

puntoEP = (**0, 116, 5008, 5618, 0, 42**) (vettore2)

```
#define N 6
int main () {
  int vettore1[N], vettore2[N];
```

Sottoproblemi?
Funzioni? ☺

Trasferimento tra array - algoritmo

vettore1 = (3259, 116, 5008, 5618, 47, 42)
→ vettore2 = (0, 116, 5008, 5618, 0, 42)

```
#define N 6
int main () {
  int vettore1[N], vettore2[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 []  ???
il tipo
dell'array
di interi …
```

Trasferimento tra array - main()

vettore1 = (**3259, 116, 5008, 5618, 47, 42**)
→ vettore2 = (**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 vettore1[N], vettore2[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 - main()

vettore1 = (**3259, 116, 5008, 5618, 47, 42**)
→ vettore2 = (**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]);
```

```
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
```

Trasferimento tra array - main()

vettore1 = (**3259, 116, 5008, 5618, 47, 42**)
→ vettore2 = (**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 vettore1[N], vettore2[N];

  printf ("caro/a utente, … dati del vettore: \n");
  leggiVettore(vettore1);
  printf ("ho letto il seguente vettore:\n");
  stampaVettore(vettore1);
  printf ("\n... hold on, transfer in progress ...\n\n");
  trasferimentoPari(vettore1, vettore2);
  printf ("dopo il trasferimento …: \n");
  stampaVettore(vettore2);
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(vettore1);
  printf ("ho letto il seguente vettore:\n");
  stampaVettore(vettore1);
  printf ("\n... hold on, transfer in progress ...\n\n");
  trasferimentoPari(vettore1, vettore2);
  printf ("dopo il trasferimento …: \n");
  stampaVettore(vettore2);
printf ("\nFINE programma\n");
```

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

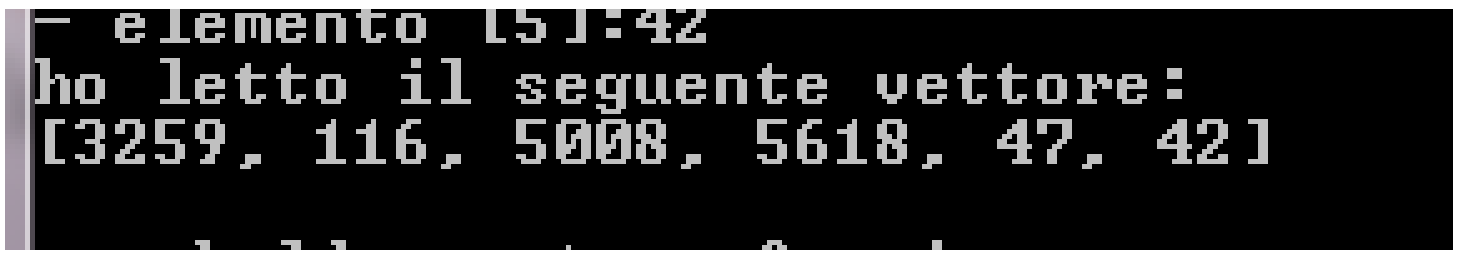
```
/* funzione di stampa di un vettore di N interi,
nel formato [num, num, num ..., num] */

                [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;
}
```



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;
}
```

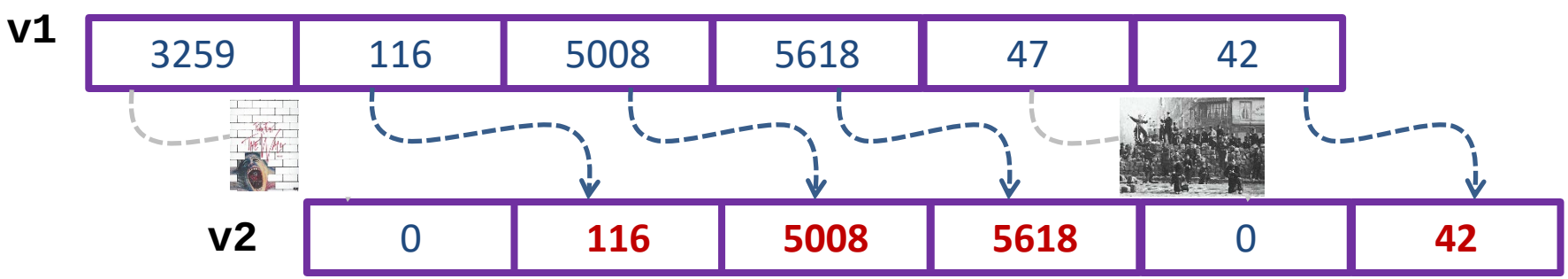
```
- elemento [5]:42
ho letto il seguente vettore:
[3259, 116, 5008, 5618, 47, 42]
```

Trasferimento tra array – funzione finale

```
/* definizione della funzione che trasferisce i pari dal
primo al secondo vettore e azzera 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;
}
```

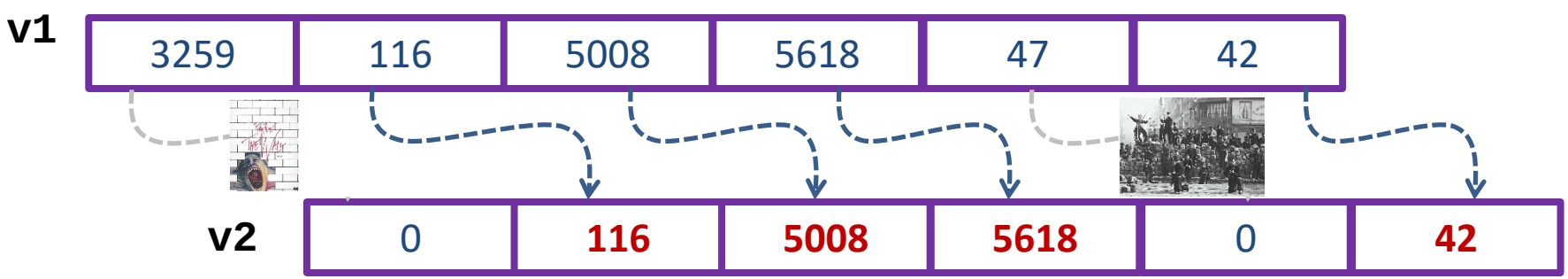


Trasferimento tra array – funzione finale

```
/* definizione della funzione che trasferisce i pari dal
primo al secondo vettore e azzera 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)
      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

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]

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

i 0

j 0

(stampa di 100 numeri su 10 righe …)

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]

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]);
```

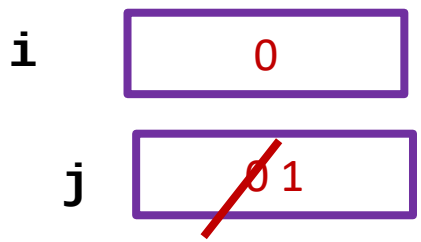
i 0

j 0

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]

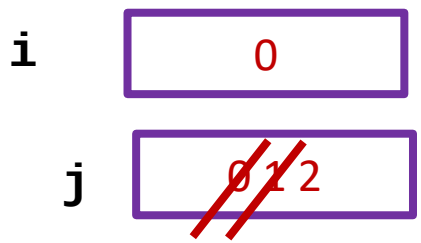



```
int mat[3][4];
…
 for (i=0; i<3; i++)
      for (j=0; j<4; j++)
          printf("mat[%d][%d]: %d\n",
                          i, j,  mat[i][j]);
```

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]);
```

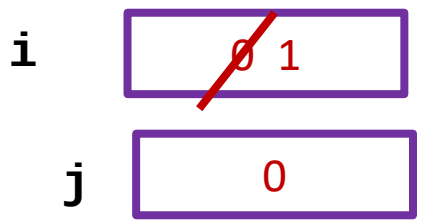
i 0

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]

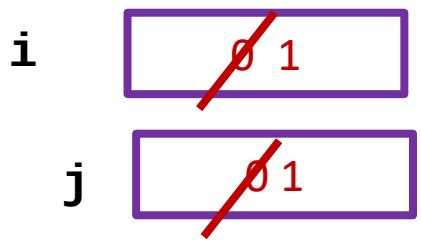



```
int mat[3][4];
…
 for (i=0; i<3; i++)
      for (j=0; j<4; j++)
          printf("mat[%d][%d]: %d\n",
                          i, j,  mat[i][j]);
```

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]

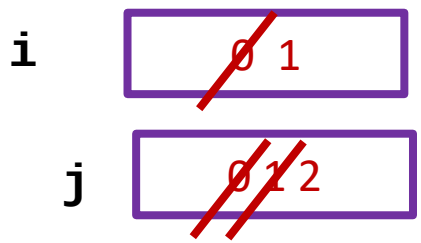



```
int mat[3][4];
…
 for (i=0; i<3; i++)
      for (j=0; j<4; j++)
          printf("mat[%d][%d]: %d\n",
                          i, j,  mat[i][j]);
```

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]

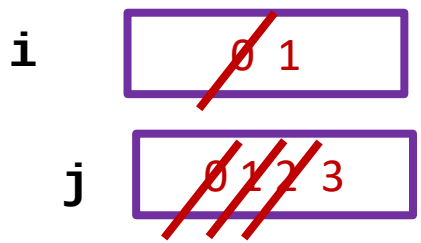



```
int mat[3][4];
…
 for (i=0; i<3; i++)
      for (j=0; j<4; j++)
          printf("mat[%d][%d]: %d\n",
                          i, j,  mat[i][j]);
```

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]



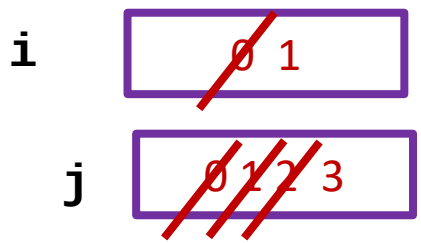

```
int mat[3][4];
…
 for (i=0; i<3; i++)
      for (j=0; j<4; j++)
          printf("mat[%d][%d]: %d\n",
                          i, j,  mat[i][j]);
```

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
```

idimensionale sono individuate da due
variare il primo indice e, per ogni suo
ondo 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

```
#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
```

Algoritmo per la stampa di una matrice (meno dettagliato … più sbrigativo …)

```
#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) per indice_riga che va da 0 a N-1
   1.1) per indice_colonna che va da 0 a M-1
        1.1.1) stampa mat[indice_riga][indice_colonna]'\n'
```

Iterazioni su array bidimensionali

Una stampa migliore

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]

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

```
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
```

i: 0 1

j: 0 1 2 3

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

A.2) E poi va a capo

Iterazioni su array bidimensionali

Una stampa migliore

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]

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');
}
```

i: 0 1 2

j: 0 1 2

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

A.2) E poi va a capo

Array bidimensionali: uso di costanti per le dimensioni

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]

```
#define N 3
#define M 4

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 … anche se nelle prossime slide usiamo 3 e 4 in esempi ad hoc … vale la pena di abituarsi ad usare le costanti per le dimensioni

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 };
```

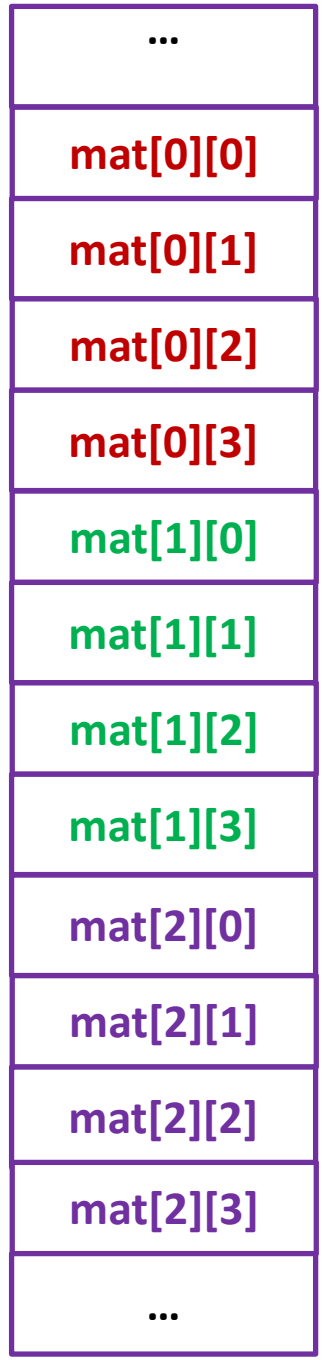
Si', ma in memoria?

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]);
```



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
```