

Elementi di Informatica

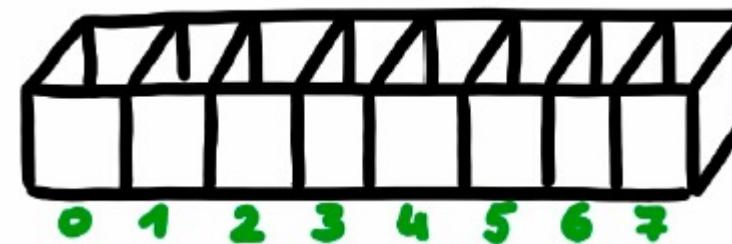
Array – Parte III

Giordano Da Lozzo e Giuseppe Sansonetti

Previously... Array

E' una sequenza di variabili omogenee.

Permette di accedere in lettura ed in scrittura a ciascuna delle variabili che lo compongono sulla base della loro posizione, detta indice.



Previously... Array: concetti ulteriori

Gli **array** (monodimensionali) sono anche detti **vettori**.

Gli **array** costituiscono una **struttura dati**, ovvero un'entità o un metodo utilizzato per **memorizzare insiemi di dati**.

- Il metodo per memorizzare dati associato ad un array è quello di memorizzare i valori degli elementi **uno dopo l'altro** (**logicamente** e vedremo **anche fisicamente**), nell'ordine in cui compaiono nell'array.

Gli **array** sono **variabili strutturate**, mentre le variabili semplici sono non strutturate.

Gli **array** hanno un **tipo**, che è il **tipo dei loro elementi**.



Previously... Dichiarazione di array

Per usare un array, è necessario dichiarare una variabile che permetta di referenziare l'array, specificando tipo, nome e lunghezza dell'array.

Esempi: `int interi[20];` `float reali[15/3];`

Sintassi: tipo nome[lunghezza];

Semantica: alloca un'area di memoria sufficiente a contenere l'array.

Nota: lunghezza deve essere una espressione di tipo int

`int interi[3.5];`



error: size of array 'interi' has non-integer type



Previously... Accesso

Dopo aver dichiarato una variabile per referenziare l'array, è possibile utilizzare l'array.

- L'unico utilizzo che si può fare di un array è accedere ai suoi elementi per memorizzare un valore al loro interno (accesso in scrittura) o per recuperare un valore già memorizzato al loro interno (accesso in lettura).

Sintassi: *nome*[*indice*]

Semantica: accede all'elemento con indice *indice* all'interno dell'array referenziato da *nome*.

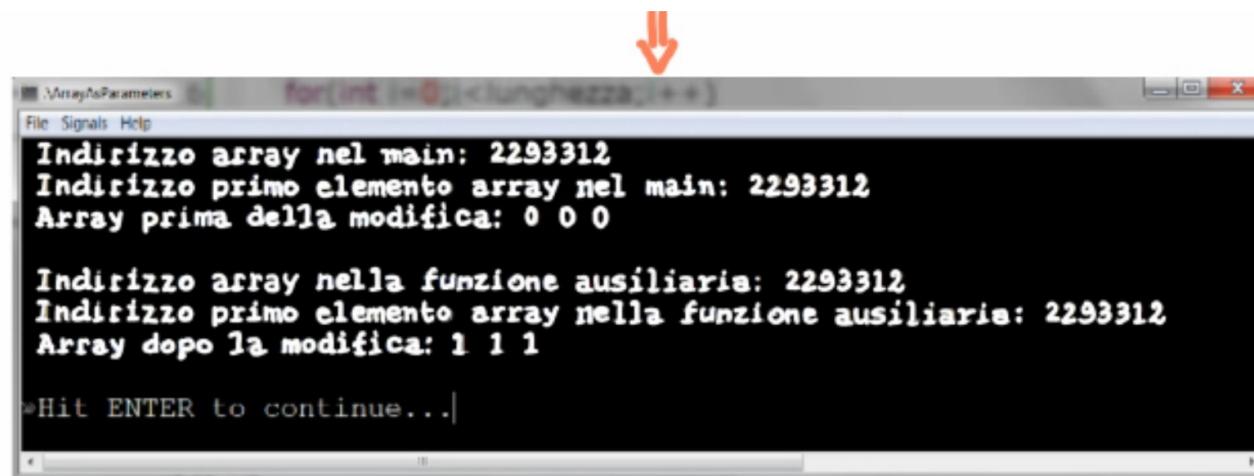
Previously... Array come parametri

Il meccanismo di legame fra parametri di tipo primitivo si chiama **per valore**: il valore del parametro attuale viene copiato nel parametro formale, che viene utilizzato come una variabile indipendente dal parametro attuale.

- Gli **array** in C non possono essere passati ad una funzione **per valore**.
- Gli **array** in C vengono passati **per riferimento** (ovvero viene passato l'**indirizzo** di memoria del **primo elemento** dell'array).
 - In realtà, viene passato per valore l'**indirizzo** del primo elemento dell'array, ma possiamo pensare che sia passato direttamente l'**indirizzo**.

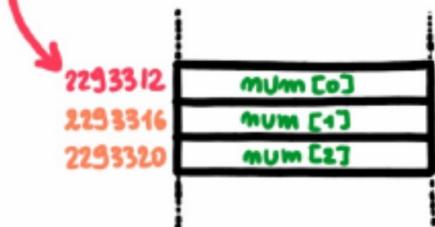
Previously... Passaggio dei parametri per riferimento

```
void incrementaElementi(int interi[], int lunghezza) {  
    printf("Indirizzo array nella funzione ausiliaria: %d\n", interi);  
    printf("Indirizzo primo elemento array nella funzione ausiliaria: %d\n", &interi[0]);  
    for(int i=0; i<lunghezza; i++)  
        interi[i]++;  
}  
  
int main() {  
    int num[3] = {0,0,0};  
    printf("Indirizzo array nel main: %d\n", num);  
    printf("Indirizzo primo elemento array nel main: %d\n", &num[0]);  
    printf("Array prima della modifica: %d %d %d\n\n", num[0], num[1], num[2]);  
    incrementaElementi(num, 3);  
    printf("Array dopo la modifica: %d %d %d\n", num[0], num[1], num[2]);  
}
```



```
ArrayParameters for: int interi[], int lunghezza, i++  
File Signals Help  
Indirizzo array nel main: 2293312  
Indirizzo primo elemento array nel main: 2293312  
Array prima della modifica: 0 0 0  
  
Indirizzo array nella funzione ausiliaria: 2293312  
Indirizzo primo elemento array nella funzione ausiliaria: 2293312  
Array dopo la modifica: 1 1 1  
  
*Hit ENTER to continue...|
```

CIO` CHE VIENE
EFFETTIVAMENTE PASSATO
DALLA FUNZIONE main
ALLA FUNZIONE
incrementaElementi È
L'INDIRIZZO DEL PRIMO
ELEMENTO DELL'ARRAY.



Previously... Quali parametri formali?

Opzione 1: *funzione(tipo nome[], int lunghezza)*

Nota: non è possibile ricavare la **lunghezza** di un array, per questo è comune che la lunghezza venga richiesta come un ulteriore parametro.

Opzione 2: *funzione(tipo *nome, int lunghezza)*

Nota: il parametro è un **puntatore**. Nelle **opzioni 1 e 2** il **parametro formale nome** «**decade**» (viene trasformato) in un puntatore.

- E' per **semplicità concettuale** che viene ammesso che un parametro formale possa avere la stessa forma di un array dichiarato localmente.

Previously... Restituire un array?

In C le funzioni **non** possono **restituire** un array!

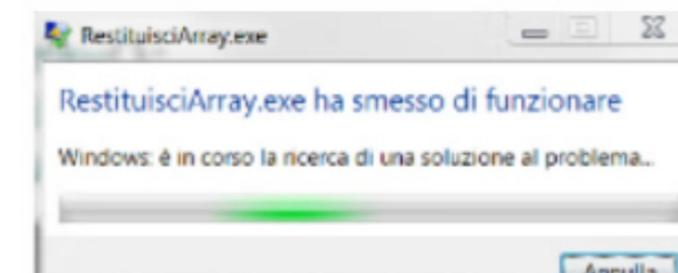
```
int[] creaArray() {  
    int sequenza[10] = {1,2,3,4,5,6,7,8,9,10};  
    return sequenza;  
}
```

```
int main(){  
    int array[10] = creaArray();  
    for(int i=0; i<10; i++)  
        printf("%d ", array[i]);  
}
```

⇒ **error: expected identifier or '(' before '[' token**

Le funzioni **possono** restituire un puntatore
ma comunque non si può utilizzare un
puntatore ad un array statico che è stato
creato in una **funzione** la cui esecuzione è
terminata, in quanto quando una funzione
termina, le sue variabili locali vengono
deallocate dalla memoria

```
int* creaArray() {  
    int sequenza[10] = {1,2,3,4,5,6,7,8,9,10};  
    return sequenza;  
}  
  
int main(){  
    int* puntatore;  
    puntatore = creaArray();  
    for(int i=0; i<10; i++)  
        printf("%d ", puntatore[i]);  
}
```



warning: function returns address of local variable [-Wreturn-local-addr]

Array bidimensionali

Un **array bidimensionale** (o **matrice**) rappresenta un **insieme** di elementi organizzati in **forma tabulare**, ovvero in **righe** e **colonne**.

Un array bidimensionale è caratterizzato da **2 dimensioni**, ovvero il **numero di righe** ed il **numero di colonne** dell'array. Il **numero di elementi** dell'array è pari al **prodotto** fra il numero di righe ed il numero di colonne dell'array.

Ogni elemento è associato a **due indici**:

- Il **primo indice** rappresenta la **riga dell'array** in cui compare l'elemento;
 - il valore di tale indice è compreso tra **0** e **numero_righe-1**.
- Il **secondo indice** rappresenta la **colonna dell'array** in cui compare l'elemento;
 - il valore di tale indice è compreso tra **0** e **numero_colonne-1**.

Array bidimensionali

Array bidimensionale in cui il numero di righe è 4 ed il numero di colonne è 5. L'array contiene quindi $4*5=20$ elementi.

Gli indici di riga variano quindi fra 0 e 3, mentre gli indici di colonna variano fra 0 e 4.

2 ^o DIMENSIONE					
0	1	2	3	4	
0	0	-3	9	5	13
1	2	-7	0	9	0
2	12	-2	4	1	-1
3	0	-5	5	17	6

L'elemento che ha valore -1 è associato con l'indice di riga 2 e con l'indice di colonna 4.

Nota: gli elementi che compongono un array bidimensionale hanno lo stesso tipo e lo stesso significato, in quanto rappresentano un insieme (in forma tabulare) di variabili omogenee.



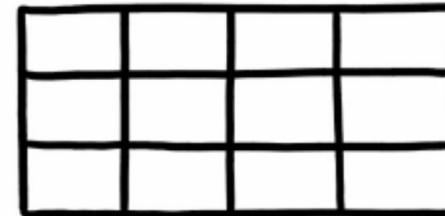
Dichiarazione di array bidimensionale

Sintassi: *tipo nome[numero_righe][numero_colonne];*

Semantica: **alloca** un'area di memoria sufficiente a contenere l'array bidimensionale. Tale area di memoria può essere referenziata tramite il nome.

int interi [3][4];

⇒



Matrice di 3 righe e 4 colonne i cui elementi sono di tipo **int**

float reali [5][2];

⇒



Matrice di 5 righe e 2 colonne i cui elementi sono di tipo **float**



Accesso

Dopo aver **dichiarato** una variabile per referenziare l'array bidimensionale, è possibile **accedere** ai suoi elementi in scrittura o in lettura.

Sintassi: *nome[indice_di_riga][indice_di_colonna]*

Semantica: accede all'elemento posizionato sulla riga *indice_di_riga* e sulla colonna *indice_di_colonna*.

interi[i][j] accede all'elemento dell'array bidimensionale *interi* che si trova sulla **riga** il cui **indice** è *i* e sulla **colonna** il cui **indice** è *j*.

ARRAY BIDIMENSIONALE *a*

<i>a[0][0]</i>	<i>a[0][1]</i>	<i>a[0][2]</i>	<i>a[0][3]</i>
<i>a[1][0]</i>	<i>a[1][1]</i>	<i>a[1][2]</i>	<i>a[1][3]</i>
<i>a[2][0]</i>	<i>a[2][1]</i>	<i>a[2][2]</i>	<i>a[2][3]</i>

Array bidimensionali e iterazioni

Gli **elementi** di un array bidimensionale vengono molto spesso **acceduti** all'interno di **istruzioni ripetitive**. Per gestire la bidimensionalità delle matrici, si usa un'**istruzione ripetitiva annidata** all'interno di un'altra **istruzione ripetitiva**, con due **variabili contatore** distinte.

```
for (i=0; i<numero_righe; i++) {  
    for (j=0; j<numero_colonne; j++) {  
        //Qui accedi all'elemento matrice[i][j]  
    }  
}
```

Per ciascun valore della variabile ***i*** fra **0** e **numero_righe-1**, la variabile ***j*** assume tutti i valori fra **0** e **numero_colonne-1**. Ovvero, per ciascun indice di riga vengono considerati tutti gli indici di colonna.

Array bidimensionali e iterazioni

In una esecuzione di  vengono acceduti tutti gli elementi della prima riga, poi tutti gli elementi della seconda riga, poi...

```
for (i=0; i<numero_righe; i++) {  
    for (j=0; j<numero_colonne; j++) {  
        //Qui accedi all'elemento matrice[i][j]  
    }  
}
```

i=0 → ESEGUI CORPO FOR ESTERNO

j=0 → accedi a matrice[0][0]

j=1 → accedi a matrice[0][1]

...

j=numero_colonne-1 → accedi a matrice[0][numero_colonne-1]

i=1 → ESEGUI CORPO FOR ESTERNO

j=0 → accedi a matrice[1][0]

j=1 → accedi a matrice[1][1]

...

j=numero_colonne-1 → accedi a matrice[1][numero_colonne-1]

...

i=numero_righe-1 → ESEGUI CORPO FOR ESTERNO

j=0 → accedi a matrice[numero_righe-1][0]

j=1 → accedi a matrice[numero_righe-1][1]

...

j=numero_colonne-1 → accedi a matrice[numero_righe-1][numero_colonne-1]

Array bidimensionali e iterazioni

```
for (i=0; i<numero_righe; i++) {  
    for (j=0; j<numero_colonne; j++) {  
        //Qui accedi all'elemento matrice[i][j]  
    }  
}
```

→ Permette di accedere a tutti gli elementi, **una riga per volta**.

```
for (j=0; j<numero_colonne; j++) {  
    for (i=0; i<numero_righe; i++) {  
        //Qui accedi all'elemento matrice[i][j]  
    }  
}
```

→ Permette di accedere a tutti gli elementi, **una colonna per volta**.

Esempio 1: matrice identità

Realizzare un programma **Matricidentita** che legge un intero n e crea e stampa la matrice identità di dimensione n .



Esempio 1: matrice identità

Realizzare un programma **Matricidentita** che legge un intero n e crea e stampa la matrice identità di dimensione n .

```
#include <stdio.h>
/* programma che legge un valore n e crea e visualizza la matrice identità di dimensione n*/
int main() {
    int n; // dimensione matrice
    /* leggi dimensione e crea matrice */
    printf("Qual e' la dimensione della tua matrice?\n");
    scanf("%d", &n);

    int matrice[n][n];

    /* riempi i valori della matrice */
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            if(i==j)
                matrice[i][j] = 1;
            else
                matrice[i][j] = 0;

    /* visualizza la matrice */
    printf("\nEcco la matrice identita' di dimensione %d:\n\n", n);
    for(int i=0; i<n; i++) {
        for(int j=0; j<n; j++)
            printf("%d ", matrice[i][j]);
        printf("\n");
    }
}
```

Esempio 2: lettura e visualizzazione

Realizzare un programma **LeggiVisualizza** che legge i valori di un array bidimensionale di tipo **int** e lo visualizza in forma matriciale, assumendo che i numeri introdotti abbiano al massimo 3 cifre.



Esempio 2: lettura e visualizzazione

Realizzare un programma **LeggiVisualizza** che legge i valori di un array bidimensionale di tipo **int** e lo visualizza in forma matriciale, assumendo che i numeri introdotti abbiano al massimo 5 cifre.

```
#include <stdio.h>
/* programma che legge i valori di un array bidimensionale di elementi di tipo int e lo visualizza in forma matriciale,
assumendo che i numeri introdotti abbiano al massimo 3 cifre */
int main() {
    int numrighe, numcolonne; // dimensioni array
    /* leggi dimensioni e crea array */
    printf("Quante righe ha la tua matrice?\n");
    scanf("%d", &numrighe);
    printf("Quante colonne ha la tua matrice?\n");
    scanf("%d", &numcolonne);

    int matrice[numrighe][numcolonne];
    /* leggi i valori della matrice, uno alla volta */
    for(int i=0; i< numrighe; i++)
        for(int j=0; j< numcolonne; j++){
            printf("Introduci l'elemento con indice di riga %d ed indice di colonna %d: ", i, j);
            scanf("%d", &matrice[i][j]);
        }
    /* visualizza la matrice, allineando a destra gli elementi */
    printf("\n");
    for(int i=0; i<numrighe; i++) {
        for(int j=0; j<numcolonne; j++)
            printf("%5d", matrice[i][j]);
        printf("\n");
    }
}
```

Esempio 3: prodotto fra matrici

Realizzare un programma `ProdottoFraMatrici` che legge 2 matrici e ne calcola il prodotto.



Esempio 3: prodotto fra matrici (1/3)

Realizzare un programma **ProdottoFraMatrici** che legge 2 matrici e ne calcola il prodotto.

```
/* Programma che legge i valori di due matrici di elementi di tipo int e
 * ne calcola e visualizza il prodotto matriciale */
int main() {
    int righe1, colonne1; // dimensioni matrice1
    int righe2, colonne2; // dimensioni matrice2

    /* leggi dimensioni e crea matrici */
    printf("Quante righe ha la matrice 1? ");
    scanf("%d", &righe1);
    printf("Quante colonne ha matrice 1? ");
    scanf("%d", &colonne1);
    int matrice1[righe1][colonne1];
    righe2 = colonne1;
    printf("La matrice 2 ha %d righe. Quante colonne ha? ", righe2);
    scanf("%d", &colonne2);
    int matrice2[righe2][colonne2];

    /* leggi i valori della matrice 1, uno alla volta */
    printf("\nRiempiamo la matrice 1!\n");
    for(int i=0; i<righe1; i++)
        for(int j=0; j<colonne1; j++){
            printf("Introduci l'elemento con indice di riga %d ed indice di colonna %d: ", i, j);
            scanf("%d", &matrice1[i][j]);
    }
```

Esempio 3: prodotto fra matrici (2/3)

```
/* leggi i valori della matrice 2, uno alla volta */
printf("\nRiempiamo la matrice 2!\n");
for(int i=0; i<righe2; i++)
    for(int j=0; j<colonne2; j++){
        printf("Introduci l'elemento con indice di riga %d ed indice di colonna %d: ", i, j);
        scanf("%d", &matrice2[i][j]);
    }

/* la matrice risultato ha dimensione righe1 x colonne2 */
int prodotto[righe1][colonne2];

/* guarda tutti gli elementi della matrice risultato */
for(int i=0; i<righe1; i++)
    for(int j=0; j<colonne2; j++){
        /* l'elemento con indici i,j è pari al prodotto scalare fra la riga i
         * di matrice1 e la colonna j di matrice2 */
        prodotto[i][j]=0;
        for(int k=0; k<righe2;k++)
            prodotto[i][j]+=matrice1[i][k]*matrice2[k][j];
    }
```

Esempio 3: prodotto fra matrici (3/3)

```
/* visualizza la matrice risultato */
printf("\nIl prodotto fra la matrice\n\n");
for(int i=0; i<righe1; i++) {
    for(int j=0; j<colonne1; j++)
        printf("%5d", matrice1[i][j]);
    printf("\n");
}
printf("\ne la matrice\n\n");
for(int i=0; i<righe2; i++) {
    for(int j=0; j<colonne2; j++)
        printf("%5d", matrice2[i][j]);
    printf("\n");
}
printf("\nvale\n\n");
for(int i=0; i<righe1; i++) {
    for(int j=0; j<colonne2; j++)
        printf("%5d", prodotto[i][j]);
    printf("\n");
}
```

Dichiarazione con inizializzazione

```
int m1[3][4] = {  
    {0,1,2,3}, // prima riga  
    {4,5,6,7}, // seconda riga  
    {8,9,10,11} // terza riga  
};
```



```
int m3[3][4] = {  
    0,1,2,3,4,5,6,7,8,9,10,11 // tutto insieme  
};
```



```
int m4[][] = {  
    {0,1,2,3}, // prima riga  
    {4,5,6,7}, // seconda riga  
    {8,9,10,11} // terza riga  
};
```



error: array type has incomplete element type 'int[]'

```
int m2[][][4] = {  
    {0,1,2,3}, // prima riga  
    {4,5,6,7}, // seconda riga  
    {8,9,10,11} // terza riga  
};
```



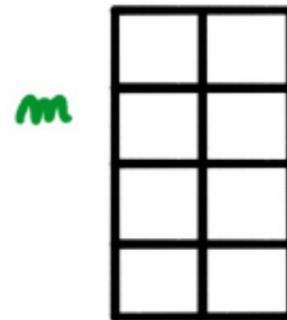
```
int m5[3][] = {  
    {0,1,2,3}, // prima riga  
    {4,5,6,7}, // seconda riga  
    {8,9,10,11} // terza riga  
};
```



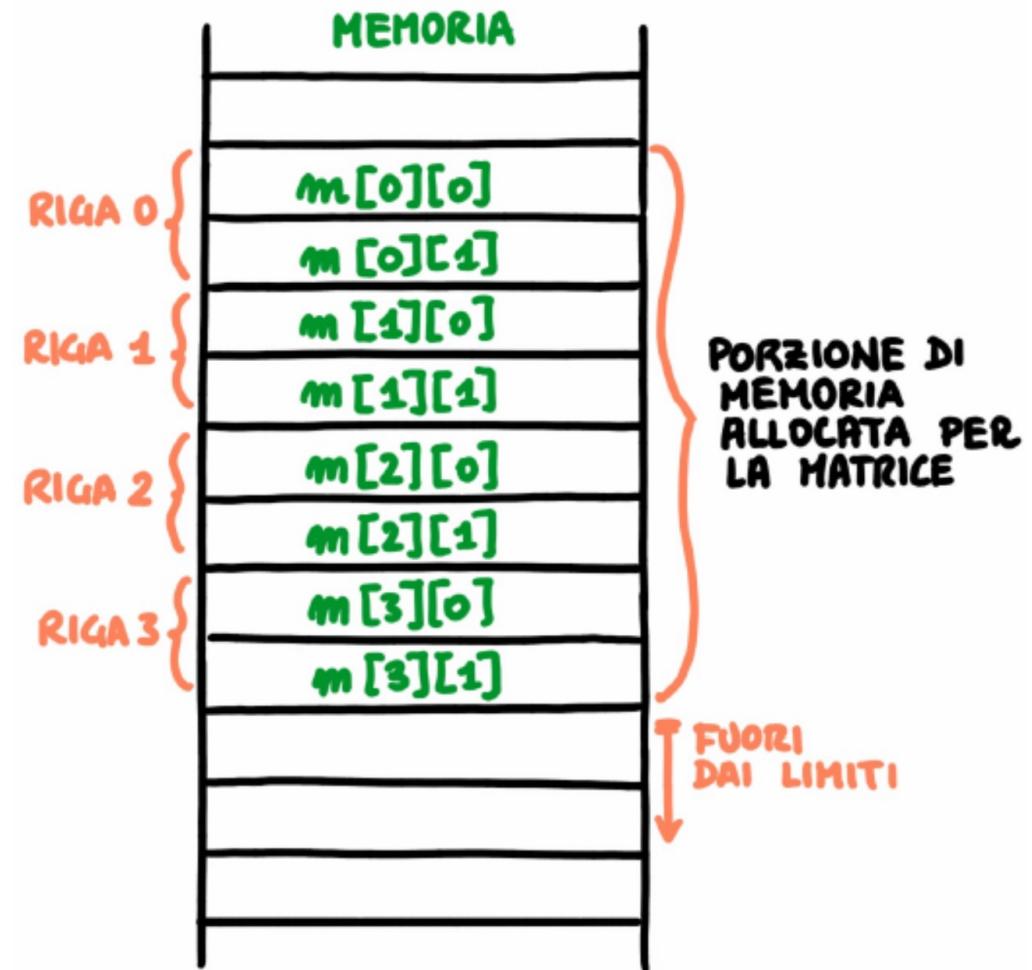
Matrici in memoria

`int m[4][2];`

Da un punto di vista **concettuale**,
questa dichiarazione crea:



Ma da un punto di vista **fisico**, gli elementi di una matrice sono rappresentati in memoria uno dopo l'altro **una riga alla volta**.



Matrici in memoria

```
int matrice[4][2] = {1,2,3,4,5,6,7,8};  
for(int i=0; i<4; i++)  
    for(int j=0; j<2; j++)  
        printf("Elemento con valore %d ed indirizzo %d\n", matrice[i][j], &matrice[i][j]);
```

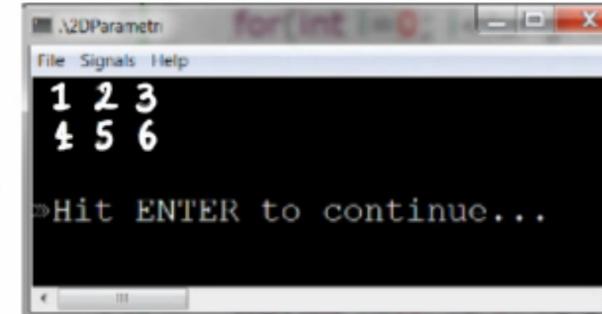


Matrici in memoria: conseguenze

E' semplice ottenere un vettore che rappresenta una riga di una matrice, scrivendo *nome_matrice[i]* se *i* è l'indice della riga richiesta.

```
/* funzione che stampa un vettore */
void stampaVettore(int riga[], int lunghezza) {
    for(int i=0; i<lunghezza; i++)
        printf("%d ", riga[i]);
}

int main() {
    int matrice[2][3]={{1,2,3},{4,5,6}};
    /* stampa ciascuna riga dell'array */
    for(int i=0; i<2; i++) {
        stampaVettore(matrice[i], 3);
        printf("\n");
    }
}
```



IL VALORE DI *matrice[i]* È L'INDIRIZZO DEL PRIMO ELEMENTO DELLA RIGA

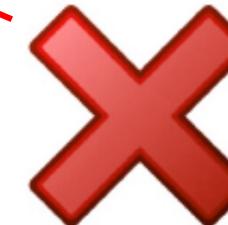
Non è possibile ottenere in maniera diretta un vettore che rappresenta una colonna della matrice!

Matrici come parametri

Se il **parametro formale** di una funzione è una **matrice**, è **necessario specificare il numero di colonne** della matrice stessa.

```
/* funzione che stampa una matrice */
void stampaMatrice(int matrice[][], int righe, int colonne) {
    for(int i=0; i<righe; i++) {
        for(int j=0; j<colonne; j++)
            printf("%d ", matrice[i][j]);
        printf("\n");
    }
}

int main() {
    int matrice[2][3]={{1,2,3},{4,5,6}};
    /* stampa la matrice */
    stampaMatrice(matrice, 2, 3);
}
error: array type has incomplete element type 'int[]'
```



```
/* funzione che stampa una matrice con 3 colonne */
void stampaMatrice(int matrice[][3], int righe) {
    for(int i=0; i<righe; i++) {
        for(int j=0; j<3; j++)
            printf("%d ", matrice[i][j]);
        printf("\n");
    }
}

int main() {
    int matrice[2][3]={{1,2,3},{4,5,6}};
    /* stampa la matrice */
    stampaMatrice(matrice, 2);
}
```



Il **decadimento (la trasformazione)** di un array a puntatore avviene **una volta**, ovvero la matrice viene trasformata in un puntatore ad un vettore (la **prima riga** della matrice)

Matrici come parametri

Come facciamo se il numero di colonne non è noto?

Non è semplice definire una funzione per gestire matrici!

Una possibilità (non conforme ad ANSI C Standard) consiste nell'«appiattire» tali matrici in vettori e gestire esplicitamente gli indici.

UN VETTORE COME PARAMETRO FORMALE

```
/* funzione che stampa una matrice */  
void stampaMatrice(int matriceAppiattita[], int righe, int colonne) {  
    for(int i=0; i<righe; i++) {  
        for(int j=0; j<colonne; j++)  
            printf("%d ", matriceAppiattita[i*colonne + j]);  
        printf("\n");  
    }  
}
```

```
int main() {  
    int matrice[2][3]={{1,2,3},{4,5,6}};  
    /* stampa la matrice */  
    stampaMatrice(&matrice[0][0], 2, 3);  
}
```

L'INDICE NEL VETTORE
DELL'ELEMENTO CHE HA INDICI i, j
NELLA MATRICE È
 $i * colonne + j$

IL PARAMETRO ATTUALE È
L'INDIRIZZO DEL PRIMO ELEMENTO
DELLA MATRICE

Altre risorse

- Bellini, Guidi: [Linguaggio C – 10.1 – 10.5](#)