Laboratorio di Programmazione

Corso di Laurea in Informatica

Gr. 3 (N-Z)

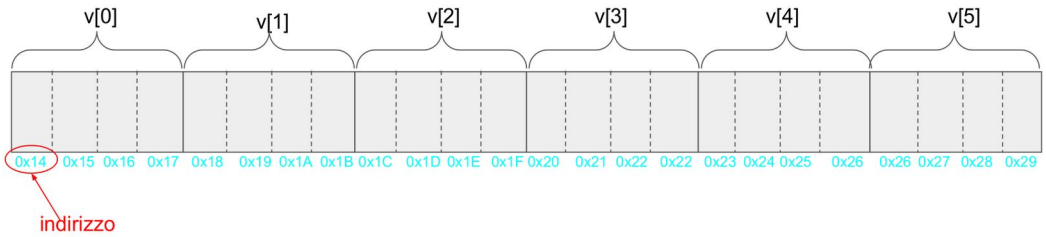
Università degli Studi di Napoli Federico II

A.A. 2022/23

A. Apicella

Array

- **sequenza** di elementi di tipo **omogeneo**
- identificato da un nome
- ogni elemento è identificato da un indice di posizione
- rappresentato in memoria come una **sequenza** di byte
- l'indirizzo di un array corrisponde all'indirizzo del **primo** elemento che lo compone



Array

```
int V[6];

V[4] = 2;
```

v[0]	v[1]	v[2]	v[3]	v[4]	v[5]
?	?	?	?	2	?

Da notare che l'operatore `[]` assume due significati differenti in base a dove è usato:

- in fase di **dichiarazione/definizione**: specifica *quanti* elementi compongono l'array
- in fase di **accesso**: specifica *dove* leggere/scrivere il dato. In tale fase è detto anche *subscript operator*.

il valore tra `[]` deve essere di tipo intero, e può essere sia un valore costante che una variabile.
Esempio:

- `int a = 4;`
- `v[a] = 42; // inserisce nella posizione di indice 4 il valore 42`

In fase di accesso i valori di posizione plausibili vanno da 0 a $n-1$ dove n è il numero di elementi che compongono l'array esplicitato in fase di dichiarazione.

Vettori

Un array può essere utilizzato in maniera naturale per rappresentare un vettore

matematica

$$\mathbf{v} = (45, 2, 7, 1, 52, 66)$$

C

```
int v[6] = {45,2,7,1,52,66};
```

memoria

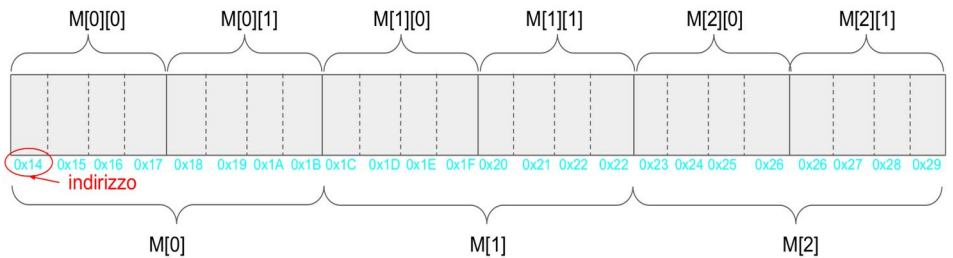
v[0]	v[1]	v[2]	v[3]	v[4]	v[5]
45	2	7	1	52	66

Array bidimensionali

- Il C permette di dichiarare array di array, ossia composto a sua volta di array dello stesso tipo
- ogni elemento di un array multidimensionale è quindi, a sua volta, un array

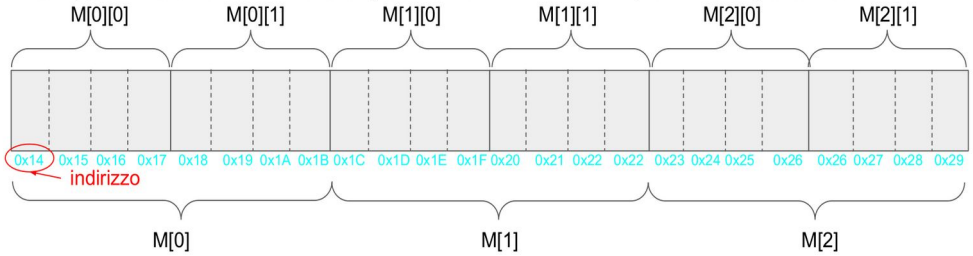
```
int M[3][2]; // dichiara un array composto da 3 array, ognuno dei quali
             // è a sua volta composto da 2 int

M[1][0] = 5; // inserisce il 5 nel secondo array (indice 1),
             // in prima posizione (indice 0)
```



Array bidimensionali

La rappresentazione in memoria di un array multidimensionale è identica a quella di un array monodimensionale

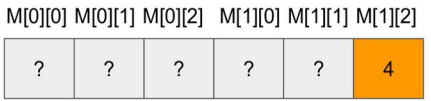


- Quando viene allocato un vettore bidimensionale, viene allocato nella realtà un array monodimensionale
- Cambia però l'indicizzazione logica, ossia gli indici necessari a leggere/scrivere un dato valore
- Internamente, l'array bidimensionale viene gestito come un array monodimensionale

Array bidimensionali

```
int M[2][3];

M[1][2] = 4;
```



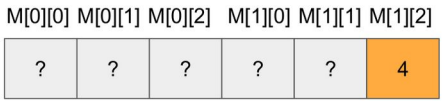
posizione reale: 5

Internamente, gli indici di posizione vengono convertiti nella posizione reale dell'array in memoria

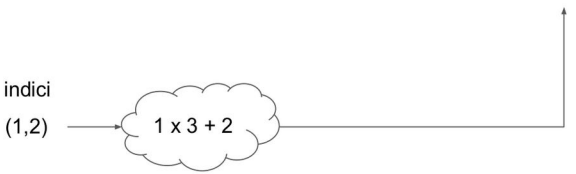
Array bidimensionali

```
int M[2][3];

M[1][2] = 4;
```



posizione reale: 5



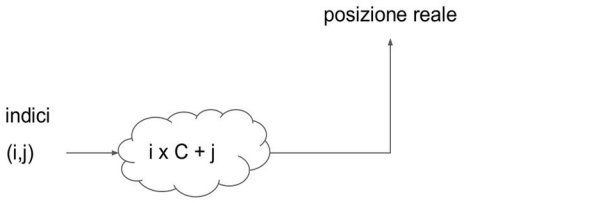
Internamente, gli indici di posizione vengono convertiti nella posizione reale dell'array in memoria

Array bidimensionali

```
int M[3][2];

M[2][1] = 4;
```

M[0][0]	M[0][1]	M[0][2]	M[1][0]	M[1][1]	M[1][2]
?	?	?	?	?	4



con C numero di elementi che compongono i sottoarray (in questo caso, 2)

Matrici

Un array bidimensionale può essere utilizzato in maniera naturale per rappresentare una matrice

matematica

$$M = \begin{pmatrix} 45 & 2 \\ 7 & 1 \\ 52 & 66 \end{pmatrix}$$

C

```
int M[3][2] = {
                {45,  2},
                { 7,  1},
                {52, 66}
              };
```

memoria

M[0][0]	M[0][1]	M[1][0]	M[1][1]	M[2][0]	M[2][1]
45	2	7	1	52	66

ogni sottovettore rappresenta una **riga** della matrice
→ il primo indice rappresenta il numero di riga dell'elemento
→ il secondo indice rappresenta il numero di colonna dell'elemento

Dato che **in C** gli elementi sono memorizzati per riga, la strategia di memorizzazione è detta **row-major order**

Row-major order

matematica *memoria*

$$M = \begin{pmatrix} 45 & 2 \\ 7 & 1 \\ 52 & 66 \end{pmatrix}$$

M[0][0]	M[0][1]	M[1][0]	M[1][1]	M[2][0]	M[2][1]
45	2	7	1	52	66

Andiamo a leggere sequenzialmente gli elementi dell'array in memoria,
e annotiamo i corrispondenti indici di matrice:

1° elemento:	(0, 0)
2° elemento:	(0, 1)
3° elemento:	(1, 0)
4° elemento:	(1, 1)
5° elemento:	(2, 0)
6° elemento:	(2, 1)

Row-major order

matematica *memoria*

$$M = \begin{pmatrix} 45 & 2 \\ 7 & 1 \\ 52 & 66 \end{pmatrix}$$

M[0][0]	M[0][1]	M[1][0]	M[1][1]	M[2][0]	M[2][1]
45	2	7	1	52	66

Andiamo a leggere sequenzialmente gli elementi dell'array in memoria,
e annotiamo i corrispondenti indici di matrice:

1° elemento:	(0, 0)
2° elemento:	(0, 1)
3° elemento:	(1, 0)
4° elemento:	(1, 1)
5° elemento:	(2, 0)
6° elemento:	(2, 1)

l'indice di colonna
varia più velocemente
dell'indice di riga

il Row-major order è la strategia di memorizzazione degli array multidimensionali
in C

Column-major order

matematica

$$M = \begin{pmatrix} 45 & 2 \\ 7 & 1 \\ 52 & 66 \end{pmatrix}$$

memoria

M[0][0]	M[1][0]	M[2][0]	M[0][1]	M[1][1]	M[2][1]
45	7	52	2	1	66

Andiamo a leggere sequenzialmente gli elementi dell'array in memoria,
e annotiamo i corrispondenti indici di matrice:

1° elemento: (0, 0)
2° elemento: (1, 0)
3° elemento: (2, 0)
4° elemento: (0, 1)
5° elemento: (1, 1)
6° elemento: (2, 1)

l'indice di riga
varia più velocemente
dell'indice di riga

il Column-major order è la strategia di memorizzazione degli array
multidimensionali in altri linguaggi, come MATLAB o FORTRAN

Row-major VS Column-major

$$M = \begin{pmatrix} 45 & 2 \\ 7 & 1 \\ 52 & 66 \end{pmatrix}$$

Row-major

M[0][0]	M[0][1]	M[1][0]	M[1][1]	M[2][0]	M[2][1]
45	2	7	1	52	66

Column-major

M[0][0]	M[1][0]	M[2][0]	M[0][1]	M[1][1]	M[2][1]
45	7	52	2	1	66

1° elemento: (0, 0) (0, 0)
2° elemento: (0, 1) (1, 0)
3° elemento: (1, 0) (2, 0)
4° elemento: (1, 1) (0, 1)
5° elemento: (2, 0) (1, 1)
6° elemento: (2, 1) (2, 1)

l'indice di colonna
varia più velocemente
dell'indice di riga

l'indice di riga
varia più velocemente
dell'indice di riga

- Se un linguaggio utilizza il **row-major order**, allora gli elementi in memoria saranno disposti in modo tale che, leggendo sequenzialmente gli elementi in memoria ed i relativi indici, la "velocità" di cambiamento degli indici aumenti *dalla prima dimensione all'ultima dimensione*.

- Se un linguaggio utilizza il **column-major order** allora gli elementi in memoria saranno disposti in modo tale che, leggendo sequenzialmente gli elementi in memoria ed i relativi indici, la "velocità" di cambiamento degli indici aumenti *dall'ultima dimensione alla prima dimensione*.

Array multidimensionali

E' possibile generalizzare ad un numero qualsiasi di dimensioni

```
int M[3][2]; // dichiara un array composto da 3 array, ognuno dei quali
             // è a sua volta composto da 2 int

float T[4][3][5]; // dichiara un array composto da 4 array,
                  // ognuno dei quali è a sua volta composto da 3 array,
                  // ognuno dei quali è composto a sua volta da 5 array
```

Array multidimensionali

```
int K[3][4];  // array di 12 elementi distribuiti lungo 2 dimensioni

int W[2][3][2]; // array di 12 elementi distribuiti lungo 3 dimensioni

int Q[1][2][2][3]; // array di 12 elementi distribuiti lungo 4 dimensioni
```

Gli **indici** di un array vengono utilizzati per identificare un elemento in fase di lettura/scrittura, esattamente come in un sistema di coordinate.

```
K[2][0]       = 5;  // assegno il valore 5 alla posizione di indici (2,0)

W[2][0][0]    = 5;  // assegno il valore 5 alla posizione di indici (2,0,0)

Q[0][2][0][0] = 5;  // assegno il valore 5 alla posizione di indici (0,2,0,0)
```

Array multidimensionali

```
int T[2][3][4] = {
                      {{1,5,7,9},
                       {2,4,6,8},
                       {5,6,2,3}},
                      {{3,1,8,4},
                       {6,2,4,7},
                       {5,9,4,1}}
                    };
```

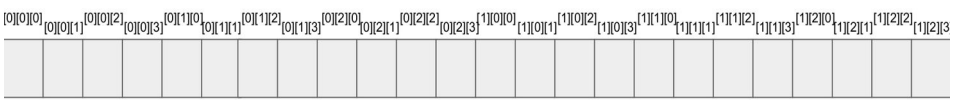
Come sono disposti in memoria?

Array multidimensionali

```
int T[2][3][4] = {
                      {{1,5,7,9},
                       {2,4,6,8},
                       {5,6,2,3}},
                      {{3,1,8,4},
                       {6,2,4,7},
                       {5,9,4,1}}
                    };
```

Come sono disposti in memoria?

Row-major order: la velocità di cambiamento aumenta dalla prima all'ultima dimensione



Array multidimensionali

```
int T[2][3][4] = {
                    {{1,5,7,9},
                     {2,4,6,8},
                     {5,6,2,3}},
                    {{3,1,8,4},
                     {6,2,4,7},
                     {5,9,4,1}}
                  };
```

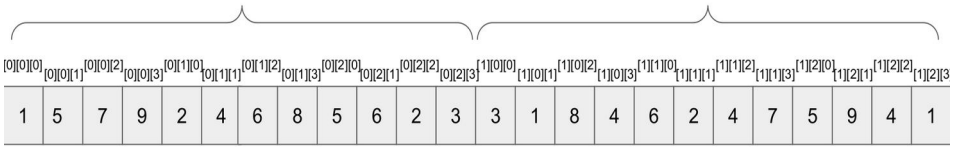
Come sono disposti in memoria?

Row-major order: la velocità di cambiamento aumenta dalla prima all'ultima dimensione

Array multidimensionali

```
int T[2][3][4] = {
                    {{1,5,7,9},
                     {2,4,6,8},
                     {5,6,2,3}},
                    {{3,1,8,4},
                     {6,2,4,7},
                     {5,9,4,1}}
                  };
```

Può essere visto come un array composto da 2 matrici 3 x 4



Array multidimensionali

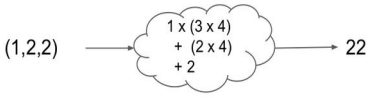
```
int T[2][3][4] = {
                        {{1,5,7,9},
                         {2,4,6,8},
                         {5,6,2,3}},
                        {{3,1,8,4},
                         {6,2,4,7},
                         {5,9,4,1}}
                       };
```

data una configurazione di indici, come calcolare la posizione reale?
Esempio:

Array multidimensionali

```
int T[2][3][4] = {
                        {{1,5,7,9},
                         {2,4,6,8},
                         {5,6,2,3}},
                        {{3,1,8,4},
                         {6,2,4,7},
                         {5,9,4,1}}
                       };
```

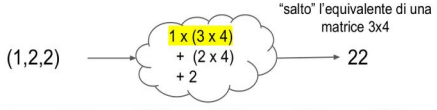
data una configurazione di indici, come calcolare la posizione reale?
Esempio:

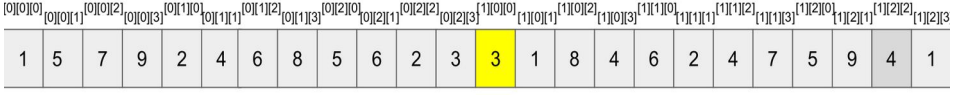


Array multidimensionali

```
int T[2][3][4] = {
                        {{1,5,7,9},
                         {2,4,6,8},
                         {5,6,2,3}},
                        {{3,1,8,4},
                         {6,2,4,7},
                         {5,9,4,1}}
                      };
```

data una configurazione di indici, come calcolare la posizione reale?
Esempio:

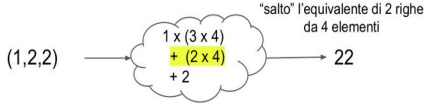


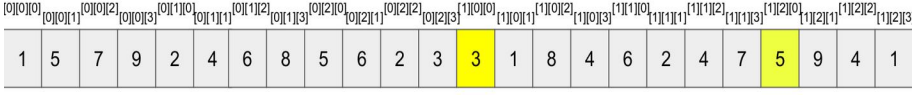


Array multidimensionali

```
int T[2][3][4] = {
                        {{1,5,7,9},
                         {2,4,6,8},
                         {5,6,2,3}},
                        {{3,1,8,4},
                         {6,2,4,7},
                         {5,9,4,1}}
                      };
```

data una configurazione di indici, come calcolare la posizione reale?
Esempio:

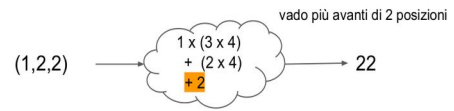


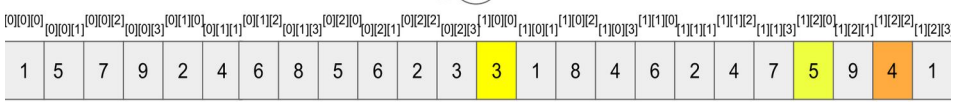


Array multidimensionali

```
int T[2][3][4] = {
                        {{1,5,7,9},
                         {2,4,6,8},
                         {5,6,2,3}},
                        {{3,1,8,4},
                         {6,2,4,7},
                         {5,9,4,1}}
                       };
```

data una configurazione di indici, come calcolare la posizione reale?
Esempio:

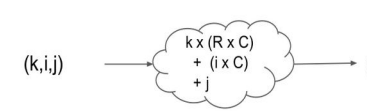


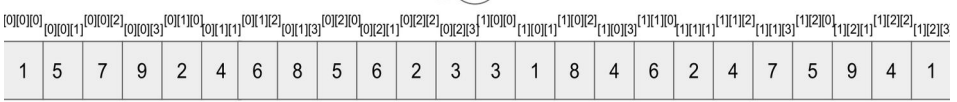


Array multidimensionali

```
int T[2][3][4] = {
                        {{1,5,7,9},
                         {2,4,6,8},
                         {5,6,2,3}},
                        {{3,1,8,4},
                         {6,2,4,7},
                         {5,9,4,1}}
                       };
```

data una configurazione di indici, come calcolare la posizione reale?
Esempio:





Array sovradimensionati

La dimensione dei vettori automatici viene decisa nel codice sorgente, e quindi fissata a *compile time*. Spesso però si vorrebbe che la dimensione effettiva di tali vettori possa essere impostata durante la materiale esecuzione del programma, ossia a *run-time*.

Esempi:

- dato un vettore di 10 elementi, estrarre gli elementi pari ed inserirli in un altro vettore
  - non si può determinare la lunghezza del secondo vettore se non dopo una scansione del primo vettore a *run time*
- acquisire dall'utente un vettore di n interi con n *deciso dall'utente*
  - Impossibile sapere la lunghezza del vettore prima del run time in cui l'utente comunica quanti elementi inserire

Array sovradimensionati

In questi casi, le possibilità sono almeno due:

- utilizzo di allocazione dinamica della memoria (vedremo poi)
- utilizzare una *stima* della lunghezza

Esempi:

- se si vuole produrre un "sottovettore", la lunghezza sarà *al più* pari alla lunghezza del vettore di partenza. Si può quindi dichiarare un nuovo vettore di uguale lunghezza, t*enendo traccia di quanti elementi saranno effettivamente utilizzati*

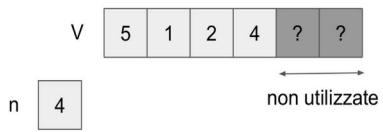
- se la dimensione è decisa dall'utente, si può fare una stima di quanti elementi un utente potrebbe inserire (e.g., non più di 100), avvisandolo opportunamente di tale limite. Si può quindi dichiarare un vettore di 100 elementi, *tenendo traccia di quanti elementi saranno effettivamente utilizzati*

Array sovradimensionati

Entrambi questi esempi fanno uso di vettori sovradimensionati → necessaria quindi una ulteriore variabile che contenga quanti siano gli elementi realmente utilizzati.

```
int main()
{
    int V[6]; // dichiaro un vettore di 6 elementi
    int n;   // variabile di supporto contenente il numero di elementi effettivi
    printf("quanti elementi vuoi inserire nel vettore? (NB: MAX 6)");
    scanf("%d",&n);

    // l'accesso al vettore avviene soltanto sui primi n elementi
    for(int i=0; i < n; i++)
    {
       scanf("%d",&V[i]);
    }
    return 0;
}
```



Array sovradimensionati

- In alcuni testi, la variabile contenente il numero di elementi effettivi del vettore è detta *riempimento*. **S**consiglio di usare questo termine.
- Lo stesso discorso si può generalizzare al caso degli array multidimensionali. Ad esempio, nel caso delle matrici, si può chiedere all'utente di fornire il numero di righe ed il numero di colonne materialmente utilizzate, a fronte di una matrice allocata sovradimensionata.
- Potreste aver sentito parlare degli Array di Lunghezza Variabile (*Variable Length Array*, VLA).
  Non saranno trattati in questo corso. Sconsigliati.