# Tipi di dato strutturati: Array

- I tipi di dato visti finora sono tutti semplici: `int`, `char`, `float`, ...
- ma i dati manipolati nelle applicazioni reali sono spesso complessi (o strutturati)
- Gli array sono uno dei tipi di dato strutturati
  - sono composti da elementi omogenei (tutti dello stesso tipo)
  - ogni elemento è identificato all'interno dell'array da un numero d'ordine detto indice dell'elemento
  - il numero di elementi dell'array è prefissato e detto lunghezza (o dimensione) dell'array
- Consentono di rappresentare tabelle, matrici, matrici n-dimensionali, ...

# Array monodimensionali (o vettori)

- Supponiamo di dover rappresentare e manipolare la classifica di un campionato cui partecipano 16 squadre.
- È del tutto naturale pensare ad una tabella

Classifica

| Squadra A | Squadra B | ... | Squadra C |
|---|---|---|---|
| $1^o$ posto | $2^o$ posto | | $16^o$ posto |

che si evolve con il procedere del campionato

Classifica

| Squadra B | Squadra A | ... | Squadra C |
|---|---|---|---|
| $1^o$ posto | $2^o$ posto | | $16^o$ posto |

**Sintassi:** dichiarazione di variabile di tipo vettore

```
tipo-elementi nome-array [lunghezza];
```

**Esempio:** `int vet[6];`

dichiara un vettore di 6 elementi, ciascuno di tipo intero.

- All'atto di questa dichiarazione vengono riservate (allocate) 6 locazioni di memoria consecutive, ciascuna contenente un intero. La lunghezza del vettore è 6.
- La lunghezza di un vettore deve essere costante (nota a tempo di compilazione). Non si può ad esempio chiedere all'utente di immettere il valore della lunghezza.
- Ogni elemento del vettore è una variabile identificata dal nome del vettore e da un indice, ad esempio `vet[i]`

  **Sintassi:** elemento di array `nome-array[espressione];`

  **Attenzione**: `espressione` deve essere di tipo intero ed il suo valore deve essere compreso tra 0 a lunghezza-1.

- **Esempio:**

| indice | elemento | variabile |
|---|---|---|
| 0 | ? | `vet[0]` |
| 1 | ? | `vet[1]` |
| 2 | ? | `vet[2]` |
| 3 | ? | `vet[3]` |
| 4 | ? | `vet[4]` |
| 5 | ? | `vet[5]` |

- `vet[i]` è l'elemento del vettore `vet` di indice `i`.
  Ogni elemento del vettore è una variabile.

```
int vet[6], a;
vet[0] = 15; // Inizializzazione implicita
a = vet[0];
vet[1] = vet[0] + a; // Inizializzazione implicita
printf("%d", vet[0] + vet[1]);
```

- `vet[0]`, `vet[1]`, ecc. sono variabili intere come tutte le altre e dunque possono stare a sinistra dell'assegnamento (es. `vet[0] = 15`), come all'interno di espressioni (es. `vet[0] + a`).
- Come detto, l'indice del vettore è un'espressione.

```
index = 2;
vet[index+1] = 23;
```

# Inizializzazione di vettori

- Gli elementi del vettore possono essere inizializzati con valori costanti (valutabili a tempo di compilazione) contestualmente alla dichiarazione del vettore. [Inizializzazione esplicita]

  **Esempio:** `int n[4] = {11, 22, 33, 44};`

- l'inizializzazione deve essere contestuale alla dichiarazione

  **Esempio:**
  ```
  int n[4];
  n = {11, 22, 33, 44};
  ```
  $\Longrightarrow$ errore!

- se i valori iniziali sono meno degli elementi, i rimanenti vengono posti a 0

  | | |
  |---|---|
  | `int n[10] = {3};` | azzera i rimanenti 9 elementi del vettore |
  | `float af[5] = {0.0};` | pone a 0.0 i 5 elementi |
  | `int x[5] = {};` | errore! |

- se ci sono più inizializzatori di elementi, si ha un errore a tempo di compilazione

  **Esempio:** `int v[2] = {1, 2, 3};` errore!

- se si mette una sequenza di valori iniziali, si può omettere la lunghezza (viene presa la lunghezza della sequenza)

  **Esempio:** `int n[] = {1, 2, 3};` equivale a `int n[3] = {1, 2, 3};`

# Altri errori tipici

- Che cosa accade se scriviamo o leggiamo un indice fuori dall'array?

```
int vet[6];
vet[6] = 15;
```

- Errore di segmentazione o Segmentation Fault a tempo di esecuzione: tentativo di accedere ad una posizione di memoria alla quale non è permesso accedere
- Che succede se leggiamo un valore da un array non inizializzato?

```
int vet[6];
x =vet[1];
```

- Il risultato della lettura non è un valore affidabile.

# Manipolazione di vettori

- avviene solitamente attraverso cicli **for**
- l'indice del ciclo varia in genere da 0 a lunghezza-1
- spesso conviene definire la lunghezza come una costante attraverso la direttiva `#define`

**Esempio:** Lettura e stampa di un vettore.

```
#include <stdio.h>
#define LUNG 5

main ()
{
int v[LUNG]; /* vettore di LUNG elementi, indicizzati da 0 a LUNG-1 */
int i;

for (i = 0; i < LUNG; i++) {
  printf("Inserisci l'elemento di indice %d:  ", i);
  scanf("%d", &v[i]);
}
printf("Indice Elemento\n");
for (i = 0; i < LUNG; i++) {
  printf("%6d %8d\n", i, v[i]); }

}
```

- In C l'unica operazione possibile sugli array è l'accesso ai singoli elementi.
- Ad esempio, non si possono effettuare direttamente delle assegnazioni tra vettori.

**Esempio:**

```
int a[3] = {11, 22, 33};
int b[3];
b = a;                          errore!
```

# Esempi

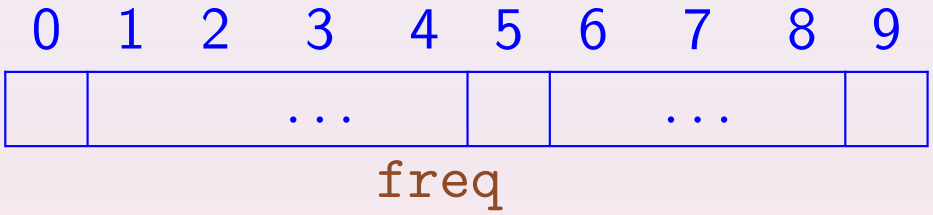
- Calcolo della somma degli elementi di un vettore.

```
int a[10], i, somma = 0;
...
for (i = 0; i < 10; i++)
  somma += a[i];
printf("%d", somma);
```

- Leggere N interi e stampare i valori maggiori di un valore intero y letto in input.

```
#include <stdio.h>
#define N 4
main()   {
int ris[N];
int y, i;
printf("Inserire i %d valori:\n", N);
for (i = 0; i < N; i++) {
  printf("Inserire valore n.  %d:  ", i+1);
  scanf("%d", &ris[i]);     }
printf("Inserire il valore y:\n");
scanf("%d", &y);
printf("Stampa i valori maggiori di %d:\n", y);
for (i = 0; i < N; i++)
  if (ris[i] > y)
    printf("L'elemento %d:  %d è maggiore di %d\n",
            i+1, ris[i],y);
}
```

- Leggere una sequenza di caratteri terminata dal carattere `\n` di fine linea e stampare le frequenze delle cifre da `'0'` a `'9'`.
- utilizziamo un vettore `freq` di 10 elementi nel quale memorizziamo le frequenze dei caratteri da `'0'` a `'9'`




`freq[0]` conta il numero di occorrenze di `'0'`

...

`freq[9]` conta il numero di occorrenze di `'9'`

- utilizziamo un ciclo per l'acquisizione dei caratteri in cui aggiorniamo una delle posizioni dell'array tutte le volte che il carattere letto è una cifra

```
int i; char ch;
int freq[10] = {0};
do {
  ch = getchar();
  switch (ch) {
    case '0':  freq[0]++; break;
    case '1':  freq[1]++; break;
    case '2':  freq[2]++; break;
    case '3':  freq[3]++; break;
    case '4':  freq[4]++; break;
    case '5':  freq[5]++; break;
    case '6':  freq[6]++; break;
    case '7':  freq[7]++; break;
    case '8':  freq[8]++; break;
    case '9':  freq[9]++; break;
    }
} while (ch != '\n');
printf("Le frequenze sono:\n");
for (i = 0; i < 10; i++)

  printf("Freq.  di %d:  %d\n", i, freq[i]);
```

- Nel ciclo **do-while**, il comando `switch` può essere rimpiazzato da un **if** come segue

```
if (ch >= '0' && ch <= '9')
  freq[ch - '0']++;
```

  Infatti:

- i codici dei caratteri da `'0'` a `'9'` sono consecutivi
- dato un carattere `ch`, l'espressione intera `ch - '0'` è la distanza del codice di `ch` dal codice del carattere `'0'`. In particolare:
  - `'0' - '0' = 0`
  - `'1' - '0' = 1`
  - ...
  - `'9' - '0' = 9`

- Leggere da tastiera e memorizzare una sequenza di interi di lunghezza prefissata DIM. Stampare quindi la sequenza in ordine inverso.

```
#include <stdio.h>
#define DIM 20
main() {
int Mem[DIM];
int i = 0;
/* inserimento dei valori */
printf("Inserire i %d interi della sequenza:\n", DIM);
for (i = 0; i < DIM; i++) {
  printf("Inserire un intero n.  %d:  ", i);
  scanf("%d", &Mem[i]); }
/* stampa in ordine inverso */
for (i = DIM -1; i > = 0; i--)
printf("%d\n", Mem[i]);
```

- Leggere da tastiera i risultati (double) di 20 esperimenti. Stampare il numero d'ordine ed il valore degli esperimenti per i quali il risultato è minore del 50% della media.

```
#include <stdio.h>
#define DIM 20
main() {
double ris[DIM], media;
int i;
/* inserimento dei valori */
printf("Inserire i %d risultati dell'esperimento:\n", DIM);
for (i = 0; i < DIM; i++) {
  printf("Inserire risultato n.  %d:  ", i);
  scanf("%lg", &ris[i]); }
/* calcolo della media */
media = 0.0;
for (i = 0; i < DIM; i++)
    media = media + ris[i];
media = media/DIM;
printf("Valore medio:  %g\n", media);
/* stampa dei valori minori di media*0.5 */
printf("Stampa dei valori minori di media*0.5:\n");
for (i = 0; i < DIM; i++)
    if (ris[i] < media * 0.5)
      printf("Risultato n.  %d:  %g\n", i, ris[i]);        }
```

# Possibili ottimizzazioni

- Calcolare la media mano a mano che si leggono i valori (si risparmia un for).
- Dichiarare una variabile soglia dove memorizzare il valore media * 0.5 da usare nei confronti, senza ogni volta doverlo calcolare di nuovo.
- ATTENZIONE: utilizzare la variabile media per fare altrettanto ci permetterebbe di risparmiare sul numero di variabili, ma renderebbe anche il codice poco leggibile e modificabile. Altrove ci potrebbe infatti servire ancora il valore originario della media.

# Array multidimensionali

**Sintassi:** dichiarazione

`tipo-elementi nome-array` $[\text{lung}_1][\text{lung}_2]\cdots[\text{lung}_n]$`;`

**Esempio:** **int** `mat[3][4];` $\Longrightarrow$ matrice $3\times4$

- Per ogni dimensione i l'indice va da 0 a $\text{lung}_i$-1.

| | | colonne 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|
| | 0 | ? | ? | ? | ? |
| righe | 1 | ? | ? | ? | ? |
| | 2 | ? | ? | ? | ? |

**Esempio:** **int** `marketing[10][5][12]`

(indici potrebbero rappresentare: prodotti, venditori, mesi dell'anno)

## Accesso agli elementi di una matrice

```
int i, mat[3][4];
...
i = mat[0][0];          elemento di riga 0 e colonna 0 (primo elemento)
mat[2][3] = 28;         elemento di riga 2 e colonna 3 (ultimo elemento)
mat[2][1] = mat[0][0] * mat[1][3];
```

- Come per i vettori, l'unica operazione possibile sulle matrici è l'accesso agli elementi tramite l'operatore `[]`.

**Esempio:** Lettura e stampa di una matrice.

```
#include <stdio.h>
#define RIG 2
#define COL 3
main()
{
int mat[RIG][COL];
int i, j;
/* lettura matrice */
printf("Lettura matrice %d x %d;\n", RIG, COL);
for (i = 0; i < RIG; i++)
  for (j = 0; j < COL; j++)
    scanf("%d", &mat[i][j]);
/* stampa matrice */
printf("La matrice è:\n");
for (i = 0; i < RIG; i++) {
   for (j = 0; j < COL; j++)
     printf("%6d ", mat[i][j]);
   printf("\n");      }   /* a capo dopo ogni riga */
}
```

**Esempio:** Programma che legge due matrici M×N (ad esempio 4×3) e calcola la matrice somma.

```
for (i = 0; i < M; i++)
  for (j = 0; j < N; j++)
    c[i][j] = a[i][j] + b[i][j];
```

## Inizializzazione di matrici

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

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |

```
int mat[2][3] = {{1,2,3}};
int mat[2][3] = {1,2,3};
```

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 0 | 0 | 0 |

```
int mat[2][3] = {{1}, {2,3}};
```

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 3 | 0 |

**Esercizio**

Programma che legge una matrice A (M×P) ed una matrice B (P×N) e calcola la matrice C prodotto di A e B

- La matrice C è di dimensione M×N.
- Il generico elemento $C_{ij}$ di C è dato da:

$$C_{ij} = \sum_{k=0}^{P-1} A_{ik} \cdot B_{kj}$$

## Soluzione

```
#define M 3
#define P 4
#define N 2
int a[M][P], b[P][N], c[M][N];
...
/* calcolo prodotto */
for (i = 0; i < M; i++)
  for (j = 0; j < N; j++) {
    c[i][j] = 0;
    for (k = 0; k < P; k++)
      c[i][j] = c[i][j] + a[i][k] * b[k][j];
  }
```

- Tutti gli elementi di `c` possono essere inizializzati a 0 al momento della dichiarazione:

```
int a[M][P], b[P][N], c[M][N] = {0};
...
for (i = 0; i < M; i++)
  for (j = 0; j < N; j++)
    for (k = 0; k < P; k++)
      c[i][j] += a[i][k] * b[k][j];
```