

Laboratorio di programmazione

Lezione IV

Tatiana Zolo

tatiana.zolo@libero.it

GLI ARRAY (MONODIMENSIONALI)

Passaggio dai tipi di dato semplice (int, float, char,...) a tipi di dato strutturato, i cui componenti sono tipi semplici o strutturati a loro volta.

Caratteristiche array:

- **elementi omogenei**: tutti gli elementi di un array sono dello stesso tipo;
- **contenitore ordinato**;
- **lunghezza dell'array (size)**: definita dal numero di elementi che contiene $\Rightarrow$ è una costante positiva (nota a tempo di compilazione);
- **indice dell'array**: numero intero compreso tra **0** e **size - 1** che individua ogni elemento dell'array.

GLI ARRAY (MONODIMENSIONALI)



Definizione: type name[size].

Possibile definire ed inizializzare simultaneamente.

Corrette:

```
int A1[10];  
float A2[4] = {6.5, 8, 4, 5.5};  
int A3[10] = {8, 1};
```

In A2 la size può essere omessa (verrebbe considerata la lunghezza della lista). In A3 i restanti 8 elementi sono posti a 0.

Errate:

```
int A4[3];  
A4 = {10, 4, -35};
```

```
int A5[4] = {1, 2, 3, 4, 5, 6, 7};
```

GLI ARRAY (MONODIMENSIONALI)

→ **Accesso**: mediante indicizzazione usando l'operatore di *subscript* '['.']
type element = name[i] (i è l'indice dell'array).
n.b.: name[0] e name[size-1] individuano rispettivamente primo ed ultimo elemento dell'array; name[size] provoca un errore (*off-by-one*).

GLI ARRAY (MONODIMENSIONALI)

- ➔ **Accesso:** mediante indicizzazione usando l'operatore di *subscript* '['']
type element = name[i] (i è l'indice dell'array).
n.b.: name[0] e name[size-1] individuano rispettivamente primo ed ultimo elemento dell'array; name[size] provoca un errore (*off-by-one*).
- ➔ **Assegnamento:** name[i] = element.
element deve essere dello stesso tipo degli elementi dell'array name.

GLI ARRAY (MONODIMENSIONALI)

- ➔ **Accesso:** mediante indicizzazione usando l'operatore di *subscript* '['']
type element = name[i] (i è l'indice dell'array).
- n.b.:** name[0] e name[size-1] individuano rispettivamente primo ed ultimo elemento dell'array; name[size] provoca un errore (*off-by-one*).
- ➔ **Assegnamento:** name[i] = element.
- element deve essere dello stesso tipo degli elementi dell'array name.
- ➔ un singolo elemento dell'array può essere utilizzato come una variabile
semplice: es. dato A un array di interi di almeno 9 elementi, è possibile scrivere `int n += A[0]*A[8]+1.`
- ➔ Non è definito un operatore per l'assegnamento tra array: è necessario assegnare individualmente ogni valore (es. con un ciclo "for").
- ➔ Non è definito un operatore per il confronto tra array: è necessario confrontare individualmente ogni valore (es. con un ciclo "for").
- ➔ Assenza di controllo dei limiti degli array, attenzione!

GLI ARRAY (MONODIMENSIONALI)

- ➔ **Accesso**: mediante indicizzazione usando l'operatore di *subscript* '['']
type element = name[i] (i è l'indice dell'array).
 - n.b.**: name[0] e name[size-1] individuano rispettivamente primo ed ultimo elemento dell'array; name[size] provoca un errore (*off-by-one*).
 - ➔ **Assegnamento**: name[i] = element.
 - element deve essere dello stesso tipo degli elementi dell'array name.
 - ➔ un singolo elemento dell'array può essere utilizzato come una variabile
semplice: es. dato A un array di interi di almeno 9 elementi, è possibile scrivere `int n += A[0]*A[8]+1`.
 - ➔ Non è definito un operatore per l'assegnamento tra array: è necessario assegnare individualmente ogni valore (es. con un ciclo "for").
 - ➔ Non è definito un operatore per il confronto tra array: è necessario confrontare individualmente ogni valore (es. con un ciclo "for").
 - ➔ Assenza di controllo dei limiti degli array, attenzione!
- es. `array_min_max.cpp`.

MATRICI (ARRAY BIDIMENSIONALI)

→ I dati sono organizzati per righe e per colonne $\Rightarrow$ per la memorizzazione si utilizza una variabile di tipo array specificando il numero di componenti per ciascuna delle due dimensioni che la costituiscono:

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

mat è una variabile strutturata che contiene 4 righe e 3 colonne, per un totale di 12 elementi.

→ Per accedere a ciascun elemento si utilizzano due indici: il primo specifica la riga, il secondo la colonna:

```
int n = mat[1][1];
```

MATRICI (ARRAY BIDIMENSIONALI)

- I dati sono organizzati per righe e per colonne $\Rightarrow$ per la memorizzazione si utilizza una variabile di tipo array specificando il numero di componenti per ciascuna delle due dimensioni che la costituiscono:

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

mat è una variabile strutturata che contiene 4 righe e 3 colonne, per un totale di 12 elementi.

- Per accedere a ciascun elemento si utilizzano due indici: il primo specifica la riga, il secondo la colonna:

```
int n = mat[1][1];
```

- In generale la dichiarazione degli **array multidimensionali** è

```
type name[dim_1][dim_2]...[dim_k];
```

Per ogni dimensione l'indice varia da 0 a $\text{dim}_i - 1$, con $i=1, \dots, k$.

MATRICI (ARRAY BIDIMENSIONALI)

- I dati sono organizzati per righe e per colonne $\Rightarrow$ per la memorizzazione si utilizza una variabile di tipo array specificando il numero di componenti per ciascuna delle due dimensioni che la costituiscono:

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

mat è una variabile strutturata che contiene 4 righe e 3 colonne, per un totale di 12 elementi.

- Per accedere a ciascun elemento si utilizzano due indici: il primo specifica la riga, il secondo la colonna:

```
int n = mat[1][1];
```

- In generale la dichiarazione degli **array multidimensionali** è

```
type name[dim_1][dim_2]...[dim_k];
```

Per ogni dimensione l'indice varia da 0 a $\text{dim}_i - 1$, con $i=1, \dots, k$.
es. `matrice.cpp`.

Esempi di inizializzazioni

→ `int M1[2,3] = {{5, 2, 8}, {4, 7, 6}};`
`int M1[2,3] = {5, 2, 8, 4, 7, 6};`

5	2	8
4	7	6

→ `int M2[2,3] = {{5, 2, 8}};`
`int M2[2,3] = {5, 2, 8};`

5	2	8
0	0	0

→ `int M3[2,3] = {{5, 2, 8}};`

5	2	0
8	0	0

Elemento della classe iostream $\Rightarrow$ stato di formattazione.
Manipolatore: **modifi ca lo stato di formattazione**. È applicato all'oggetto stream come un dato, ma anziché provocare la lettura o scrittura di dati, esso modifi ca lo stato interno dell'oggetto stream.

Elemento della classe iostream $\Rightarrow$ stato di formattazione.

Manipolatore: **modifi ca lo stato di formattazione**. È applicato

all'oggetto stream come un dato, ma anziché provocare la lettura o scrittura di dati, esso modifi ca lo stato interno dell'oggetto stream.

→ Un valore in virgola mobile per default ha una precisione di 6 cifre: per modificarla `precision(int)` oppure `setprecision()` (è necessario

il file header `<iomanip>`).

Elemento della classe iostream $\Rightarrow$ stato di formattazione.

Manipolatore: **modifi ca lo stato di formattazione**. È applicato

all'oggetto stream come un dato, ma anziché provocare la lettura o scrittura di dati, esso modifi ca lo stato interno dell'oggetto stream.

➔ Un valore in virgola mobile per default ha una precisione di 6 cifre: per modificarla `precision(int)` oppure `setprecision()` (è necessario il file header `<omanip>`).

➔ Un valore in virgola mobile per default è mostrato in notazione decimale fissa: per cambiare la visualizzazione nella notazione scientifica si usa `scientific`; per tornare alla visualizzazione decimale si usa `fixed`.

Elemento della classe iostream $\Rightarrow$ stato di formattazione.

Manipolatore: **modifi ca lo stato di formattazione**. È applicato

all'oggetto stream come un dato, ma anziché provocare la lettura o scrittura di dati, esso modifi ca lo stato interno dell'oggetto stream.

→ Un valore in virgola mobile per default ha una precisione di 6 cifre: per modificarla `precision(int)` oppure `setprecision()` (è necessario il file header `<omanip>`).

→ Un valore in virgola mobile per default è mostrato in notazione decimale fissa: per cambiare la visualizzazione nella notazione scientifica si usa `scientific`; per tornare alla visualizzazione decimale si usa `fixed`. Si può controllare la larghezza di un valore numerico o di una stringa per l'output con `width(int)` oppure `setw()` (è necessario il file header `<iomanip>`). Al contrario di tutti gli altri modifi catori, non modifi ca lo

stato di formattazione dell'oggetto stream.

Elemento della classe `iostream` $\Rightarrow$ stato di formattazione.

Manipolatore: **modifi ca lo stato di formattazione**. È applicato

all'oggetto `stream` come un dato, ma anziché provocare la lettura o scrittura di dati, esso modifi ca lo stato interno dell'oggetto `stream`.

→ Un valore in virgola mobile per default ha una precisione di 6 cifre: per

modificarla `precision(int)` oppure `setprecision()` (è necessario

il file header `<iomanip>`).

→ Un valore in virgola mobile per default è mostrato in notazione decimale

fissa: per cambiare la visualizzazione nella notazione scientifica si usa

`scientific`; per tornare alla visualizzazione decimale si usa `fixed`.

→ Si può controllare la larghezza di un valore numerico o di una stringa

per l'output con `width(int)` oppure `setw()` (è necessario il file

header `iomanip`). Al contrario di tutti gli altri modificatori, non modifi ca lo

stato di formattazione dell'oggetto `stream`.

→ es. `manip.cpp`.

Esercizi

- 1. Array:** leggi da standard input una sequenza di "n" numeri interi, con "n" dato di input. Calcolare la media e restituire la sequenza corrispondente traslata a media 0 (`media_array.cpp`).
es. Supponiamo l'utente abbia inserito 3 elementi nell'array: 4, 2, 3. Con una operazione (la stessa applicata ad ogni elemento dell'array) si ottiene il corrispondente array traslato a media 0: 1, -1, 0.
- 2. Array:** carica i punteggi (da 1 a 10) di 2 prove effettuate da 3 concorrenti e determina la classifica sapendo che il punteggio totale di ogni concorrente è dato dalla media aritmetica delle due prove. Si visualizzano con una tabella sia i risultati parziali che il punteggio finale di ogni concorrente (`classifica.cpp`).
- 3. Matrici:** calcola la trasposta di una matrice le cui dimensioni ed elementi sono inseriti dall'utente (`trasposta.cpp`).