18. Array

Corso di Algoritmi e Linguaggi di Programmazione Python/C

Angelo Cardellicchio
angelo.cardellicchio@stiima.cnr.it

Outline

- Definizione ed inizializzazione di un array
- L’operatore `[]`
- Array multidimensionali
- L’operatore **`sizeof`**
  - Ed alcuni esercizi
- Array e stringhe

Definizione ed inizializzazione di un array

- Sono delle **strutture dati** contenenti dati dello stesso tipo (in C)
- Ogni variabile dell'array è un **elemento** dello stesso, ed è contraddistinta da un **indice**
  - *In C, un array avente n elementi è caratterizzato da degli indici che vanno da 0 ad $n-1$*
- Il numero di elementi (ovvero la **dimensione**) è predefinito ed invariabile
  - *Modificare il numero di elementi di un array significa in realtà crearne uno nuovo!*
- Per definirlo ed inizializzarlo, si utilizza l'operatore [ ]

L’operatore [ ] (1)

- L’operatore `[ ]` viene usato per definire ed inizializzare un array
- Supponiamo di voler dichiarare un array contenente n numeri interi, con $n = 5$. Per farlo, useremo la seguente notazione:

```
int mio_array[5];
```

- La dichiarazione è analoga quindi a quella di una variabile
  - *Specificando per l’operatore `[ ]`, indichiamo al compilatore che non si tratta di una variabile di tipo intero, ma di un array di variabili di tipo intero di **n** elementi!*
- L’inizializzazione di un array può avvenire in maniera contestuale alla dichiarazione, come per tutte le altre variabili

L’operatore [ ] (2)

- Una notazione di questo tipo invece non è valida:

```
mio_array = {1,2,3,4,5};
```

  - *Ciò è legato al fatto che un array non è un l-value!*
- Per dichiarare ed inizializzare un array, invece:

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

- In quest’ultimo caso, è possibile omettere la dimensione dell’array, che sarà automaticamente dedotta dal compilatore:

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

- L’operatore `[]` ha anche un ruolo nella scrittura e lettura dei singoli elementi dell’array

L’operatore [ ] (3)

- L’operatore [ ] ha anche un ruolo nella scrittura e lettura dei singoli elementi dell’array
- Può essere usato per **assegnare** (scrivere) un valore all’i-mo elemento

```
mio_array[3] = 10;
// mio_array sarà ora [1, 2, 3, 10, 5]
```

- Può essere usato per **accedere** (leggere) al valore del j-mo elemento

```
int a = mio_array[2];
// a varrà 3
```

Array multidimesionali [1]

- È possibile creare degli array ad n dimensioni
  - *Un esempio di array a due dimensioni è una matrice!*
- Per farlo, si utilizza una notazione di questo tipo:

```
float matrice [3][3];
```

- È possibile estendere questa notazione ad un numero arbitrario di dimensioni
- Valgono le stesse regole usate per gli array monodimensionali
- Occorre tenere conto che gli elementi sono memorizzati con gli indici meno significativi a destra

Array multidimesionali [2]

- Ad esempio, per definire un array bidimensionale:

```
int matrice [3][3] = { {2, 0, 1}, {1, 3, 2}, {4, 3, 3} };
```

- L'idea è quindi di definire un **array di array**
- L'ordine in cui sono memorizzati gli elementi è il seguente:

```
matrice [0][0]; matrice [0][1]; matrice [0][2];
matrice [1][0]; ...; matrice [2][2];
```

L’operatore `sizeof` (ed alcuni esercizi)

- L’operatore **`sizeof`** restituisce il numero di byte complessivi dell’array
  - *Questo dipende dal numero di elementi dell’array e dal tipo dello stesso!*
- ***Esercizio 1***: *un tensore è un array ad n dimensioni contenente valori arbitrari. Creare due tensori di dimensioni $3 \times 3 \times 3$, uno contenente valori interi, e l’altro contenente valori in formato `double`. Usare l’operatore sizeof per confrontarne lo spazio occupato in memoria, e visualizzare a schermo tutti i valori dell’array più ‘pesante’.*

Array e stringhe (1)

- Si definisce **stringa** una **sequenza di caratteri**
- Non sono un tipo primitivo nel C; tuttavia, sono molto utilizzate
  - *Esempio: la* **`printf`** *accetta come argomento una stringa!*
- In memoria, una stringa è rappresentata come un array di **`char` null terminated**
  - *Ciò significa che l'elemento dell'array che segue l'ultimo carattere della stringa deve essere* **`null`**
- È possibile usare un numero di caratteri *inferiore* alla lunghezza dell'array, ma *mai* superiore

Array e stringhe (2)

- Una stringa può quindi essere inizializzata come un array…
  - *…a condizione che l'ultimo valore sia* **`null`***!*
  - *L'escape character associato a* **`null`** *è* **`\0`***.*

```
char stringa[10] = { ‘C’, ‘i’, ‘a’, ‘o’, ‘\0’ };
```

- Esiste un modo più semplice per inizializzare la stringa…
  - *…ovvero usare le doppie apici!*

```
char stringa[10] = "Ciao"
```

- In questo caso, il terminatore sarà aggiunto automaticamente
- **Nota: non è possibile usare le stringhe come l-value in un'assegnazione!**

Array e stringhe (3)

- Non è possibile convertire direttamente una stringa (ovvero un array) in un numero
- Per farlo, occorre usare le funzioni **`atoi`** ed **`atof`**, definite nell'header **`stdlib.h`**

```
atoi("10")      // Da stringa ad intero; restituisce 10
atof("1.1")     // Da stringa a float; restituisce 1.1
```

- La funzione duale è la **`sprintf`**, analoga alla **`printf`**, ma che accetta un ulteriore parametro, ovvero la stringa da dare in output
  - *Altra differenza è che la printf manda il risultato sullo «standard output», ovvero sulla riga di comando, mentre la sprintf restituisce una stringa*

```
sprintf(stringa, "Formatto l'intero %d", 10);
```

- L'istruzione precedente restituirà la stringa ***Formatto l'intero 10***.

Array e stringhe (4)

- ***Esercizio 2***: scriviamo un programma che, data in ingresso una stringa rappresentativa di un numero x, con x numero reale o naturale, chiami l'adeguata funzione per convertirlo in una variabile di tipo numerico. Utilizziamo poi il risultato restituito dalla funzione **`sprintf`** per visualizzare a schermo il valore della stringa associata ad x.

Domande?

42