Introduzione al Linguaggio C

Arrays

Daniele Pighin

March 2009

Outline

Outline

Cosa é e a cosa serve un array

- Gli array sono sequenze di dimensione finita di variabili omogenee
- Ciascun elemento di un array é una variabile
- Tutti gli elementi di un array sono rappresentati contiguamente in memoria
- La sua dimensione (numero di elementi) é definita in fase di dichiarazione
- La sua dimensione non cambia durante la sua vita
- E' possibile definire array uni-dimensionali (vettori) e bi-dimensionali (matrici)
- Ciascun elemento puó essere acceduto in tempo $O(1)$ mediante il suo indice
- Il primo elemento di un array di dimensione n ha indice 0, l'ultimo $n-1$
- Le funzioni non possono ritornare array!

Outline

Creazione di un array

```
1  int    a[5];  // dichiara un array di 5 variabili intere
2
3  float  b[2];  // dichiara un array di 2 variabili float,
4                // ad esempio un punto in R^2
5
6  char   c[10]; // dichiara un array di 10 caratteri
7
8  double d[] = {10, 5, 0.5} // dichiara un array di double di
9                            // lunghezza 3 e ne inizializza
10                           // i valori
11
12 int matrice1[4][2];  // dichiara una matrice di interi con
13                      // 4 righe e due colonne
14
15 int matrice2[2][2] = { {10, 4}, {20, 5}}; /* dichiara una
16      matrice con due righe e due colonne e ne inizializza
17      i valori. La prima riga contiene {10, 4} e la seconda
18      {20, 5}. Ciascuna riga di una matrice é un array
19     monodimensionale */
```

- L'accesso in lettura/scrittura agli elementi di un array passa attraverso gli indici dell'array

```
1  int myarray[5]; // crea un array di lunghezza 5
2
3  myarray[0] = 10; //imposta il valore del primo elemento
4  myarray[4] = 20; //imposta valore dell'ultimo elemento
5
6  myarray[5] = 5; // errore! L'elemento di indice 5 non
7                  // é definito!
8
9  int i;
10 for (i=0; i<5; i++) {
11     myarray[i] = (i+1)*(i+1); // che cosa fa?
12 }
```

Outline

Passaggio di array unidimensionali

- Un array non porta con sé informazioni sulla sua lunghezza
- Ogni volta che passiamo un array ad una funzione, dobbiamo passarne anche la lunghezza o il numero di elementi su cui intendiamo lavorare ($\leq$ lunghezza!!)

```
1  int somma(int valori[], int lunghezza) {
2      int result = 0, i;
3      for (i=0; i<lunghezza, i++) {
4          result += valori[i];
5      }
6      return result;
7  }
8  int main() {
9      int myarray[] = {1, 23, 54, 7, 123};
10     int tutti = somma(myarray, 5); // somma tutti
11     int pochi = somma(myarray, 3); // somma i primi 3
12     int male = somma(myarray, 10); // errore! non ci
13                                    // sono 10 elementi
14 }
```

Passaggio di array unidimensionali (cont.)

- In C, quando passiamo variabili ad una funzione generalmente lo facciamo per valore
- La funzione crea una copia locale del valore attuale della variabile e le assegna un nome (il nome del parametro formale)
- Gli array sono invece passati per riferimento
- La funzione non lavora su una copia dell'array ma sull'array stesso!

```
1  void func(int data[], int len) {
2      int i;
3      for (i=0; i<len; i++) {
4          data[i] += 5;
5      }
6  }
7
8  int myarray[] = {3, 5, 10};
9  func(myarray, 2); // myarray ora contiene {8, 10, 10} !!
10 printf("%d", myarray[0]); // stampa 8
```

Passaggio in sola lettura: const

- Se un parametro formale ha il modificatore const, il valore della variabile corrispondente non puó essere modificato (accesso esclusivamente in lettura)
- Il compilatore produce un messaggio di errore se all'interno della funzione cerchiamo di modificarne il valore (accesso in scrittura)

```
1  void func1(const int data[], int len) {
2      int i;
3      for (i=0; i<len; i++) {
4          printf("%d\n", data[i]); // OK, accesso in lettura
5      }
6  }
7
8  void func2(const int data[], int len) {
9      int i;
10     for (i=0; i<len; i++) {
11         data[i] += 5; // ERRORE!! data é const!
12     }
13 }
```

Osservazioni

- In C gli array sono intrinsecamente legati al concetto di puntatore
- Per capire veramente come funzionano gli array (e per sfruttarli a dovere) bisogna capire i puntatori
- L'utilizzo dei puntatori (e dell'allocazione dinamica della memoria) consentono un uso molto piú flessibile degli array
- Lo vedremo prossimamente...

Outline

Esercizi

- Scrivere una funzione

  `float media(double x[], int len);`

  che ritorni la media aritmetica dei primi `len` elementi del vettore `x`.

- Scrivere una funzione

  `float max(double x[], int len);`

  che ritorni il massimo tra i primi `len` elementi di `x`.

Esercizi (cont.)

- Scrivere una funzione

  `float media_armonica(double x[], int len);`

  che ritorni la media armonica dei primi `len` elementi del vettore `x`. La media armonica di una sequenza $X = X_1, \ldots, X_n$ di n valori é definita come:

$$arm(X) = \frac{1}{\sum_{i=1}^{n} \frac{1}{X_i}}$$

Esercizi (cont.)

- Scrivere una funzione

  `float somma_quadrati(double x[], int len);`

  che ritorni la somma dei quadrati dei primi `len` elementi dell'array `x`.

Esercizi (cont.)

- Scrivere una funzione

  `void prodcoord(double a[], double b[], double c[], int n);`

  che modifichi i valori di `c` in modo che, per ogni $i = 0, \ldots, n-1$, valga: `c[i] == a[i] * b[i]`.

- Scrivere una funzione

  `void scalare(double a[], double b[], int n);`

  che ritorni il prodotto scalare degli array `a` e `b`.