

Array

Andrea Marin

Università Ca' Foscari Venezia
Laurea in Informatica
Corso di Programmazione

a.a. 2012/2013

Motivazioni

- ▶ **Problema:** Acquisire da standard input 100 valori `int` e ristamparli in ordine crescente
- ▶ **Problema:** Acquisire da standard input una sequenza di caratteri che termini con *invio* e ristamparla sostituendo le lettere minuscole con le lettere maiuscole
- ▶ Quante variabili servono per risolvere questi problemi?



Array

Definizione (Array in C)

Un array è una variabile che fa riferimento a più elementi dello stesso tipo; ogni elemento dell'array è esso stesso una variabile e può essere richiamata mediante un indice; la dimensione dell'array deve essere specificata al momento della dichiarazione e deve essere una costante.



Ulteriori indicazioni

- ▶ Ad esempio un array di `int` è un insieme indicizzato di interi
- ▶ Le locazioni di memoria che sono associate ad un array sono **contigue**

- ▶ Dichiarazione di un array in C:
`tipo nome_array[dimensione]`

Dove:

- ▶ `tipo`: è il tipo di dato delle celle dell'array
- ▶ `nome`: è l'identificatore associato all'array
- ▶ `dimensione`: è un **costante** che definisce la dimensione dell'array



Esempi di dichiarazione ed inizializzazione

Esempi di dichiarazione:

- ▶ `int array[100];`
- ▶ `double mio_vettore[50];`

Inizializzazione di array:

- ▶ Come le variabili, anche gli array possono essere inizializzati
- ▶ Esempio:

```
int vect[] = { 4, 2, 10, 22, 112, 96};
```

- ▶ Attenzione quello che precede **non** è un assegnamento ma una inizializzazione contestuale alla dichiarazione
- ▶ `vect` non è un left-value (non può comparire a sinistra di un assegnamento)

Accesso agli elementi di un array

- ▶ Ad ogni elemento di un array è associato un indice
- ▶ In un array di N elementi, gli indici vanno da 0 a $N - 1$
- ▶ Per individuare un elemento di un array si usa la notazione:

`nome_array[indice]`

Dove:

- ▶ `nome_array`: è l'identificatore associato all'array
- ▶ `indice`: è una **espressione** di tipo `int` che identifica una posizione valida (tra 0 e $N - 1$) nell'array
- ▶ Nota: C non effettua controlli run-time sulla validità dell'indice



Esempio

```
int mio_array[]={5, 3, 7, 7};  
int altro_array[3];
```

mio_array

5	3	7	7
---	---	---	---

altro_array

?	?	?
---	---	---

```
altro_array[2]=mio_array[0]+1;
```

altro_array

?	?	6
---	---	---

← 5+1



Left-value e Right-value

- ▶ Esempio: `vett[2+i]`
- ▶ L'elemento individuato mediante l'indice identifica una cella di memoria pertanto...
 - ▶ Se compare in un'espressione s'intende il valore contenuto in quella cella
 - ▶ `vett[i+2] = vett[0] + 5;`
 - ▶ Se compare a sinistra di un assegnamento, denota la locazione di memoria in cui scrivere
 - ▶ `vett[i+2] = 12;`



Conseguenze

- ▶ L'operatore di assegnamento = non effettua una copia elemento per elemento degli array
- ▶ L'operatore di confronto == non effettua un confronto elemento per elemento degli array
- ▶ L'acquisizione di un array va fatta elemento per elemento



Una questione di eleganza

- ▶ La dimensione di un array deve essere una costante intera positiva
 - ▶ `double miovet[100];`
- ▶ La leggibilità del codice viene molto migliorata dall'introduzione di costanti
 - ▶ Per introdurre una costante in C si può usare la direttiva *`#define`*
 - ▶ `#define DIM 100`
 - ▶ Si noti che manca il `;` finale
- ▶ Definizione di array:
 - ▶ `int mio_array[DIM];`
- ▶ Il compilatore rimpiazzerà ogni occorrenza di DIM con il valore 100
- ▶ Usare le maiuscole per le costanti
- ▶ Reminder: per noi il buono stile è obbligatorio



Pattern: scorrimento completo di un array

- ▶ Dal momento che le dimensioni dell'array sono note, per scorrere un array interamente il ciclo preferenziale è il `for`
- ▶ Data la seguente dichiarazione
 - ▶ `int vect [DIM];`
- ▶ Per scorrere ciascun elemento dell'array:

```
int i; /*indice per lo scorrimento*/
```

```
for (i=0; i < DIM; i = i+1) { /*i scorre da 0 a DIM-1*/  
    Operazione sull'elemento i-esimo dell'array  
}
```



Stampa tutti gli elementi di un array

```
#include <stdio.h>

#define DIM 10

double vett[] = {1.4, 5.2, 2.0, 5.1, 10.9,
                 1.1, 0.1, 33.33, -12.2, 1.12};
int i; /*Indice di scorrimento*/
int main(){

    for (i=0; i < DIM; i = i+1) {
        printf("In posizione %d si trova\n il valore %f", i, vett[i]);
    }
    return 0;
}
```



Remark

- ▶ L'indice è sempre una **espressione** di tipo `unsigned int` (intero senza segno)
- ▶ Il tipo della variabile indicizzata mediante `vett[i]` si desume dalla dichiarazione del vettore
- ▶ Nell'esempio precedente...
 - ▶ `i` è una espressione e ha tipo `int`
 - ▶ `vett[i]` è una variabile e ha tipo `float`
- ▶ Domanda per il pubblico da casa: qual è la differenza tra espressione e variable?
 - ▶ Quando una variabile è una espressione?



Esempio: sommare tutti gli elementi di un array

```
#include <stdio.h>

#define DIM 10

double vett[] = {1.4, 5.2, 2.0, 5.1, 10.9,
                 1.1, 0.1, 33.33, -12.2, 1.12};
int i; /*Indice di scorrimento*/
double somma; /*Accumulatore*/
int main(){
    somma = 0.0; /*inizializzazione acc.*/
    for (i=0; i < DIM; i = i+1) {
        somma = somma + vett[i];
    }
    printf("La somma vale %f \n", somma);
    return 0;
}
```



Cosa stampa? /1

```
#include <stdio.h>

#define DIM 10

int vett[] = {6, 4, 5, 2, 13,
              -11, -7, 44, -12, 0};
int i;  /*Indice di scorrimento*/

int main(){
    for (i=0; i < DIM; i = i+1) {
        if (i%2 == 0)
            printf("%d", vett[i]);
    }
    return 0;
}
```

- Riscrivere l'esempio senza usare il condizionale all'interno del ciclo

Cosa stampa? /2

```
#include <stdio.h>

#define DIM 10

int vett[] = {6, 4, 5, 2, 13,
              -11, -7, 44, -12, 0};
int i;  /*Indice di scorrimento*/

int main(){
    for (i=0; i < DIM; i = i+1) {
        if (vett[i]%2 == 0)
            printf('%d', vett[i]);
    }
    return 0;
}
```



Riempire un array di 1000 interi con numeri casuali tra 0 e 99

```
#include <stdio.h>
#include <stdlib.h>

#define DIM 1000

int miovett[DIM];
int i;
int main() {
    randomize(); /*inizializza il generatore*/
    for (i=0; i < DIM; i++)
        miovett[i] = random(100);

    ...
    return 0;
}
```



Proprietà universale

- ▶ Esempi:
 - ▶ controllare se un array di interi contiene solo multipli di 3
 - ▶ controllare se un array è ordinato
 - ▶ Perché è una proprietà universale?
- ▶ Il metodo è sempre lo stesso: si assume che la proprietà sia soddisfatta e si cerca un controesempio
 - ▶ Se se ne trova almeno uno allora la proprietà è falsa (e il ciclo può essere interrotto)
 - ▶ Se si scorre l'array senza trovare alcun controesempio allora la proprietà è vera



Dire se un array è ordinato in modo crescente in senso lato

```
#include <stdio.h>

#define DIM 100

int vett[DIM];
int ordinato, i;

int main() {
    /* Riempi array */
    ordinato = 1; /* assumo proprietà soddisfatta */
    i = 1; /* indice */
    while (i < DIM && ordinato) {
        if (vett[i-1] > vett[i])
            ordinato = 0;
        i = i + 1;
    }

    if (ordinato)
        printf("'L'array e' ordinato'");
    else
        printf("'L'array non e' ordinato'");

    return 0;
}
```



Variazioni sul tema...

- ▶ Perchè l'indice comincia da 1 e non da 0?
- ▶ L'algoritmo funziona anche con array di dimensione 1?
- ▶ Riscrivere l'algoritmo in modo da:
 - ▶ Non aggiungere variabili o cambiare nome
 - ▶ Dare la risposta **senza** effettuare un testo sulla variabile ordinato



Proprietà esistenziale

- ▶ Dato un array di float dire se almeno un elemento è negativo
- ▶ Applicazione del pattern per verificare una proprietà esistenziale
- ▶ Ovviamente si può riformulare come proprietà universale
 - ▶ Dato un array dire se tutti gli elementi sono positivi o nulli
 - ▶ Se si fa questo passaggio va specificato come commento nel codice



Soluzione

```
#include<stdio.h>

#define DIM 100

float vett[DIM];
int i;
int almeno_uno_neg;

int main(){
    /*carica array*/

    /*proprieta assunta falsa*/
    almeno_uno_neg = 0;

    i = 0;
    while (!almeno_uno_neg && i < DIM) {
        if (vett[i] < 0)
            almeno_uno_neg = 1;
        i = i+1;
    }
    if (almeno_uno_neg)
        printf('Presente almeno un elemento negativo');
    else
        printf('Nessun elemento negativo');
    return 0;
}
```



Nuovo pattern: trovare max/min in un array

- ▶ Esempio: dato un array di float con almeno un elemento, determinare il valore massimo contenuto
- ▶ Idea risolutiva:
 - ▶ Assumo inizialmente che il primo elemento sia il massimo e ne salvo il valore in una variabile `max`
 - ▶ `max` conterrà il valore massimo dell'array contenuto nelle posizioni tra 0 e i incluse
 - ▶ **Invariante del ciclo!**
 - ▶ i scorre tra 1 e $DIM - 1$
- ▶ Al termine del ciclo la variabile `max` contiene il valore massimo dell'array tra le posizioni 0 e $DIM - 1$, cioè di tutto l'array



Soluzione

```
#include <stdio.h>

#define DIM 100

float vett[DIM];
int i;
float max;

int main() {
    /* carica array */
    max = vett[0];

    for (i=1; i<DIM; i=i+1)
        if (vett[i] > max)
            max = vett[i];

    printf('Il valore massimo e' %d \n', max);

    return 0;
}
```



Variazioni

- ▶ Perchè l'array deve avere almeno un elemento?
- ▶ Perchè usare il ciclo `for`?
- ▶ Modificare il pattern in modo da restituire la posizione dell'elemento massimo dell'array



Array di array

- ▶ Oltre a poter definire array con elementi di tipo semplice (float, int, ...) C consente anche la definizione di array di array
- ▶ Questo consente di trattare array multidimensionali
- ▶ Esempio:
 - ▶ `double matrice[DIM][DIM];`
 - ▶ Definisce una matrice quadrata di dimensione $DIM \times DIM$
 - ▶ `int prova [] [] = {{4, 4, 2}, {5, 7, 1}};`
 - ▶ Definisce una matrice 2×3
 - ▶ `char arraybig[DIM][DIM][DIM];`
 - ▶ Array tridimensionale



Convenzioni sulle matrici bidimensionali

- ▶ Sono presenti due dimensioni
- ▶ Il primo indice denota la riga della matrice
- ▶ Il secondo indice denota la colonna
 - ▶ Attenzione: diverso dal piano Cartesiano!



Stampare gli elementi di una matrice per riga

```
#include <stdio.h>
#define DIM1 50
#define DIM2 20

int r, c;

double matrice[DIM1][DIM2];
/*matrice con 50 righe e 20 colonne*/

int main() {
    /*inizializza matrice*/

    for (r = 0; r < DIM1; r = r+1)
        for (c = 0; c < DIM2; c = c+1)
            printf("%d", matrice[r][c]);

    return 0;
}
```



Esercizi

- ▶ Stampare gli elementi di una matrice in modo da andare a capo al termine di ogni riga
- ▶ Contare gli elementi di una matrice di double che sono positivi
- ▶ Dato un array di dimensione DIM e una matrice di dimensione DIM×DIM, sovrascrivere la diagonale della matrice in modo da copiarvi gli elementi del vettore. Mantenere l'ordine, cioè l' i -mo elemento della diagonale della matrice deve essere l' i -mo elemento dell'array
- ▶ Data una matrice di float DIM1×DIM2 e un array di float di dimensione DIM1, sommare gli elementi delle righe della matrice e salvare il valore nel vettore. La posizione i del vettore deve contenere la somma degli elementi della riga i della matrice