

# Introduzione al linguaggio C

## Dati aggregati

Violetta Lonati

Università degli studi di Milano  
Dipartimento di Informatica

Laboratorio di algoritmi e strutture dati  
Corso di laurea in Informatica

# Argomenti

Dati aggregati: array, stringhe, strutture, enumerazioni

Array

Stringhe

Strutture

Enumerazioni

# Array unidimensionali

## Dichiarazione

```
#define N 100  
int a[N + 2];
```

## Indicizzazione

`a[i]` indica l'*i*-esimo elemento dell'array `a`. Al posto di `i` si può mettere una qualsiasi espressione intera.

Attenzione agli effetti collaterali in `a[i] = b[i++]`;

## Inizializzazione

```
int a[4] = {1, 2, 0, 0};  
int b[] = {1, 2, 0, 0};  
int c[4] = {1, 2};
```

Eventuali elementi mancanti sono inizializzati a 0.

# Array unidimensionali - continua

## Lunghezza di un array

```
sizeof( a ) / sizeof( a[0] )
```

## Array costanti

```
const int MESI[12] = {31,28,31,30,31,30,31,31,30,31,30,31}
```

Il qualificatore `const` rende ogni elemento imm modificabile.

## Esempi

```
for ( i = 0; i < N; i++ )  
    a[i] = 0                /* azzero l'array a */
```

```
for ( i = 0; i < N; i++ )  
    scanf( "%d", &a[i] );  /* leggo e mem. in a */
```

```
for ( i = 0; i < N; i++ )  
    printf( "%d_\n", a[i] ); /* stampo */
```

## Array unidimensionali - errori tipici

### La dimensione degli array deve essere costante

Secondo l'ansi C, la dimensione dell'array va fissata in fase di compilazione (non di esecuzione) usando un'espressione costante.

```
/* compilando con gcc -pedantic produce warning! */  
int n;  
int a[n];
```

### Non c'è controllo dei limiti

```
int a[N], i;  
for ( i = 0; i <= N; i++ )  
    printf( "%d\n", a[i] );  
/* accede ad a[N] che non e' definito!! */
```

## Array unidimensionali - esercizi

- ▶ Rovescia: scrivete un programma che legga una sequenza di numeri interi terminata da 0 e li stampi dall'ultimo (0 escluso) al primo. Potete assumere che la sequenza contenga al più 100 numeri non nulli.
- ▶ Cifre ripetute: scrivete un programma che legga in input un numero intero  $n$  usando `scanf( "%d", &n )` e stabilisca se  $n$  contiene qualche cifra ripetuta e in caso affermativo quali.
- ▶ Da base 10 a base  $b$ : scrivere un programma che data una coppia di numeri interi  $b$  e  $n$  (separati da spazio e in base 10) stampi la rappresentazione di  $n$  in base  $b$ . Potete assumere che il numero di cifre in base  $b$  sia sempre minore di 100.
- ▶ Da base  $b$  a base 10: scrivere un programma che dato un numero  $b$  (in base 10) e una sequenza  $s$  di cifre in  $\{0, \dots, b-1\}$  terminata da un punto, stampi il numero la cui rappresentazione in base  $b$  è data da  $s$ . Potete assumere che il numero di cifre di  $s$  sia sempre minore di 100.

## Array bidimensionali

```
int mat[R][C], r, c;
/* inizializzo a 0 */
for ( r = 0; r < R; r++ )
    for ( c = 0; c < C; c++ )
        mat[r][c] = 0;
```

### Mappa di memorizzazione

Un array di  $R$  righe e  $C$  colonne viene memorizzato come array di  $R * C$  elementi. In altre parole  $M[i][j]$  si trova  $i * C + j$  posizioni dopo  $M[0][0]$ .

### Inizializzazione

```
int a[3][2] = { {1, 2}, {3, 4}, {0, 0} };
int b[][2] = { {1, 2}, {3, 4}, {0, 0} };
/* non si puo' omettere la seconda dimensione! */
int c[3][2] = {1, 2, 3, 4, 0, 0};
int d[3][2] = {1, 2, 3, 4};
/* eventuali el. mancanti sono inizializzati a 0 */
```

## Array bidimensionali - esercizi

- ▶ Esami e studenti: scrivete un programma che permetta di inserire gli esiti di 5 esami per 5 studenti e calcoli la media di ciascuno studente e la media dei voti ottenuti in ciascun esame.
- ▶ Quadrato magico: scrivete un programma che legga un intero  $n$  e stampi un quadrato magico di dimensione  $n$ .

# Stringhe

Le stringhe in C sono array di char, terminati dal carattere di *fine stringa*. Il carattere di fine stringa ha valore 0 (i char sono trattati come interi!), ma è opportuno indicarlo come carattere `'\0'`. La lunghezza di una stringa è data dal numero dei suoi caratteri + 1 (per il simbolo di fine stringa).

## Inizializzazione

```
char parola1[5] = "ciao";
char parola2[] = "ciao";
char parola3[] = { 'c', 'i', 'a', 'o', '\0' };
```

## Lettura e scrittura di stringhe

```
scanf( "%s", parola ); /* senza & !!! */
printf( "%s\n", parola );
```

## Funzioni per elaborare stringhe

<string.h> contiene le dichiarazioni di alcune funzioni utili: `strcpy`, `strcmp`, `strcat`, `strlen`,...

## Stringhe - esempio

```
/* copia parola in bis*/  
char parola[] = "ciao", bis[4+1];  
int i = 0;  
  
while ( parola[i] != '\0' ) {  
    bis[i] = parola[i];  
    i++;  
}  
  
bis[i] = '\0';  
printf( "%s\n", bis );
```

## Stringhe - esercizio

Una stringa si dice *palindroma* se è uguale quando viene letta da destra a sinistra e da sinistra a destra. Quindi "enne" è palindroma, ma "papa" non lo è. Scrivete un programma che legga una stringa terminata da '.' e stabilisca se è palindroma. Potete assumere che la stringa sia al più di 100 caratteri.

# Strutture

- ▶ Si tratta di tipi di dati aggregati. Ogni variabile strutturata è composta da **membri**, ciascuno dei quali è individuato da un nome.
- ▶ Ogni struttura ha un **namespace** distinto, quindi i nomi dei membri possono essere usati anche per altre cose!
- ▶ In altri linguaggi, le strutture sono chiamate **record** e i membri **campi**.

```
/* Dichiarazione della var. strutturata studente */  
/* avente 3 membri: cognome, nome, anno. */  
struct {  
    char cognome[10];  
    char nome[10];  
    int anno;  
} studente;
```

# Strutture - continua

## Inizializzazione

```
struct {
    char cognome[10];
    char nome[10];
    int anno;
} studente1 = { "Ferrari", "Mario", 1988 },
  studente2 = { "Rossi", "Maria", 1988 };
```

## Accesso ai membri

```
studente.anno = 1987;
printf( "%s\n", studente.nome );
```

## Assegnamento

E' possibile farlo tra due variabili strutturate dello stesso tipo: tutti i membri vengono **copiati** (con gli array questo non si può fare!!).

```
studente1 = studente2;
```

## Definizione di tipi strutturati

Ci sono due modi per definire tipi strutturati:

```
/* usando un tag */
struct studente {
    char nome[10];
    char cognome[10];
    int anno;
};
...
struct studente stud1, stud2;
```

```
/* usando typedef */
typedef struct {
    char nome[10];
    char cognome[10];
    int anno;
} Studente;
...
Studente stud1, stud2;
```

## Strutture nidificate

```
typedef struct {  
    float x, y;  
} Punto;
```

```
typedef struct {  
    Punto p1, p2;  
} Rettangolo;
```

...

```
Rettangolo r;  
r.p1.x = 0;  
r.p1.y = 1;  
r.p2.x = 5;  
r.p2.y = 3;
```

**Esercizio:** Scrivete un programma che calcoli l'area e il perimetro di rettangoli e cerchi.

## Array di strutture

```
#define LUNG 20 /* lungh. massima per le stringhe */
#define N 100 /* numero massimo di voci */

typedef char Stringa[LUNG + 1];

typedef struct {
    Stringa nominativo;
    Stringa tel;
} Voce;

...

/* dichiaro rubrica come array di N voci */
Voce rubrica[N];
```

**Esercizio:** Scrivete un programma per la gestione di una semplice rubrica. Attraverso un menu l'utente deve poter visualizzare la rubrica e inserire nuovi numeri.

# Enumerazioni

- ▶ Si tratta di tipi di dati i cui valori sono *enumerati* dal programmatore.

```
enum { CUORI , QUADRI , FIORI , PICCHE }  
      seme1 , seme2 ;
```

- ▶ Ad ogni possibile valore si associa una **costante di enumerazione** intera. Per default i valori associati alle costanti sono 0, 1, 2, ..., in questo ordine. E' possibile scegliere altri valori.

```
enum { VENDITE = 10 ,  
      RICERCA = 21 ,  
      PRODOTTI = 17 } settore ;
```

- ▶ Si possono definire tipi enumerativi usando sia typedef che tag.

```
typedef enum { FALSO , VERO } Bool ;
```