

Strutture in C

I dati strutturati o definiti dall'utente

Nel linguaggio C è possibile definire tipi di dati detti strutturati attraverso la parola chiave **struct** aggregando più dati anche se di tipo diverso in un unico contenitore.

Una Struct:

- È formata da una o più variabili
- Contiene variabili di tipi diversi
- Riunisce più informazioni correlate
- Aiuta ad organizzare dati complessi

Le strutture

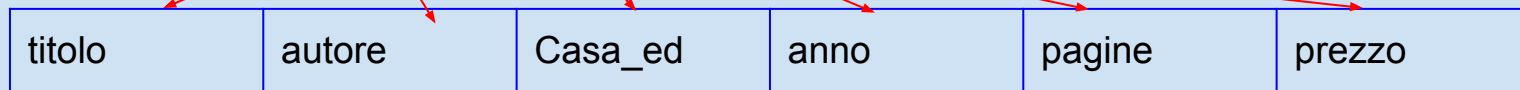
Se ad esempio volessimo raccogliere tutte le informazioni riguardanti un libro potremmo scrivere:

```
struct libro {  
    char titolo[100];  
    char autore[50];  
    char casa_ed[100];  
    int anno, pagine;  
    double prezzo;  
};
```

Le strutture

Con il codice scritto precedentemente:

- non è ancora creata nessuna variabile
- è stato solo definito un tipo di dato aggregato
- per dichiarare variabili di questo tipo:
- **struct libro L1,L2;**
- L1 e L2 sono istanze con la forma libro
- compilatore alloca memoria per tutti i
- membri (elementi, campi)



Record

Le strutture

L1 ed **L2** rappresentano il riferimento alle celle di memoria dove sono contenute le informazioni delle strutture memorizzate come sequenze di byte

L1

titolo (100 byte)

autore (50 byte)

casa_ed (100 byte)

anno (4 byte)

pagine (4 byte)

prezzo (8 byte)

L2

titolo (100 byte)

autore (50 byte)

casa_ed (100 byte)

anno (4 byte)

pagine (4 byte)

prezzo (8 byte)

Le strutture

Si possono dichiarare istanze anche così:

```
struct libro {  
    char titolo[100];  
    char autore[50];  
    char casa_ed[100];  
    int anno, pagine;  
    double prezzo;  
} L1,L2;
```

Inizializzazione dei campi

I campi di una struct possono essere inizializzati all'atto della dichiarazioni delle istanze, ovviamente bisogna assegnare i valori rispettando ordine e tipi di dati

```
struct libro l1,l2={
```

```
    "I Malavoglia",
```

```
    "Giovanni Verga",
```

```
    "Einaudi",
```

```
    2001,
```

```
    169,
```

```
    9.8};
```

In questo esempio stiamo
facendo 2 istanze della struct libro
di nome **l1** ed **l2**.

La seconda istanza è inizializzata
con i valori riportati.

Accedere agli elementi

- Per accedere agli elementi di una struct si usa la cosiddetta notazione puntata ovvero mediante l'operatore ".":

nome_variabile.nome_campo

- Si può usare in qualsiasi contesto come se fosse una semplice variabile:

```
printf("Titolo: %s",l1.titolo );
```

```
gets(l1.titolo);
```


Accedere agli elementi

- Per accedere agli elementi si usa l'operatore ".":

```
strcpy(l1.titolo,"Le ore");
```

```
strcpy(l1.autore,"M. Cunningham");
```

```
strcpy(l1.casa_ed,"Bompiani");
```

```
l1.anno=2007;
```

```
l1.pagine=149;
```

```
l1.prezzo=6.8;
```

Strutture innestate

Come già annunciato precedentemente tra gli elementi di una struttura può essere contenuta un'altra struttura basta che questa sia stata dichiarata prima

```
struct point {  
    double x,y;  
};  
  
struct triangolo {  
    struct point p1,p2,p3;  
};  
  
struct triangolo tr1,tr2;
```



Strutture innestate

- L'accesso alle componenti della sottostruttura avviene ancora mediante notazione puntata. Avremo tanti punti per quanti sono i livelli di innesti.

Per esempio:

- per accedere alla coordinata x del secondo angolo del **tr1**:

tr1.p2.x

Vettori di strutture

La cosa veramente interessante di questi nuovi tipi di dati è che avendo dichiarato le caratteristiche di un libro se poi ho 1000 libri basta semplicemente dichiarare un vettore del tipo libro con numerosità 1000

➤ un vettore di strutture:

```
struct libro {  
    char titolo[100];  
    char autore[50];  
    char casa_ed[100];  
    int anno, pagine;  
    double prezzo;  
} libri[1000];
```

E poi accedere ad ogni libro con la stessa sintassi dei vettori cioè mediante un indice, aggiungendo la notazione puntata. Ad esempio:

Libri[20].titolo

Rappresenta il titolo del libro di indice 20

Vettori di strutture

E così come per i vettori con un semplice ciclo for possiamo, stampare ad esempio, le informazioni che ci interessano

- Si accede ai campi con la notazione puntata:

```
for(i1=0;i1<1000;i1++){  
    printf("%s\n",libri[i1].titolo);  
    printf("%s\n",libri[i1].autore);  
    printf("Prezzo: %.2f Euro\n",libri[i1].prezzo);  
}
```

Vettori di strutture

Le regole di prima valgono ovviamente anche per le strutture innestate

- se la struttura contiene un vettore:

```
struct point {  
    double x[2];
```

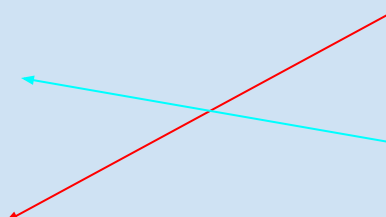
```
};
```

```
struct triangolo {  
    struct point p[3];
```

```
};
```

```
struct triangolo tr[N];
```

Stiamo dichiarando un vettore di triangoli dove ogni triangolo contiene un vettore di punti che sono i vertici



Vettori di strutture

```
for(i1=0;i1<N;i1++){           //Per ogni triangolo i1  
    printf("Triangolo n. %d:\n",i1);  
    for(i2=0;i2<3;i2++)        //Per ogni vertice i2  
        printf(" (%.2f,%.2f)\n", tr[i1].p[i2].x[0], tr[i1].p[i2].x[1]);  
}
```

Come si vede dal codice precedente ancora una volta mediante la notazione puntata e 2 cicli for possiamo stampare per tutti i triangoli i tre vertici che li rappresentano

Gli assegnamenti di strutture

Un aspetto interessante delle strutture è che è permesso l'assegnamento. In questo modo tutti i membri possono essere copiati usando un'unico assegnamento:

```
struct libro l1,l2;  
strcpy(l1.titolo,"Le ore");  
strcpy(l1.autore,"Michael Cunningham");  
strcpy(l1.casa_ed,"Bompiani");  
l1.anno=2001;  
l1.pagine=169;  
l1.prezzo=6.80;  
l2=l1; //l2 conterrà la copia di tutti i campi di l1
```


Gli assegnamenti di strutture

- anche i vettori vengono copiati

```
struct vettore {  
    double v[MAX];  
    int lunghezza;  
} v1;  
struct vettore v2= {{1.0, 2.2, 4.5}, 3}
```

Gli assegnamenti di strutture

- l'assegnamento

`v1=v2;`

è consentito

- l'assegnamento

`v1.v=v2.v;`

non è consentito

Infatti in questo caso vengono automaticamente copiati tutti i campi di **`v2`** in **`v1`** anche se come campi vi fossero dei vettori o delle stringhe

In questo caso invece non è possibile copiare tutti gli elementi del secondo vettore nel primo mediante semplice assegnamento. Per ottenerlo dovremmo fare un ciclo di copia di ogni singolo valore

Strutture e funzioni

- tre modi di passare elementi di strutture:
 - passare separatamente un membro
 - passare un'intera struttura
 - ~~● passare un puntatore ad una struttura~~

Strutture e funzioni

- si passa un membro solo quando non c'è bisogno di tutta la struttura:

```
double euro_to_dollaro(double e){
```

```
    return 1.49*e;
```

```
}
```

```
...
```

```
printf(" Prezzo: %.2f $", euro_to_dollaro(l2.prezzo));
```

Strutture e funzioni

- passare l'indirizzo di un membro

```
printf("Titolo: ");
```

```
gets(l1.titolo);
```

```
...
```

```
printf("Anno: ");
```

```
scanf("%d", &l1.anno);
```

Strutture e funzioni

- passaggio di un'intera struttura:chiamata per valore

In questo caso viene creata nella funzione mediante parametro formale una copia di tutti i campi della struttura rendendo così molto pesante il passaggio e ovviamente le modifiche alla copia non ricadranno sull'originale

```
void stampa(struct libro l){  
    printf("%s\n",l.titolo);  
    printf("%s\n",l.autore);  
    printf("Anno: %d\n",l.anno);  
    printf("Prezzo: %.2f Euro\n",l.prezzo);  
}
```

La variabile **l** crea una copia di tutti i membri della struct

Strutture e funzioni

- la dichiarazione della struttura libro deve essere globale
- devono corrispondere i nomi:

struct libro l1;

...

stampa(l1);

- non si può chiamare stampa con una struttura uguale alla struttura **libro** ma di nome diverso

Strutture e funzioni

Ovviamente una funzione può anche ritornare una struttura sia come valore che come indirizzo

```
struct point{  
    double x,y;  
};  
struct point somma(struct point p1,struct point p2){  
    struct point s;  
    s.x=p1.x+p2.x;  
    s.y=p1.y+p2.y;  
    return s;  
}
```

In questo caso viene ritornato una struttura per valore e cioè una copia di tutti i campi

Definire nuovi tipi

il C consente di definire nuovi tipi mediante la parola chiave typedef.

Esempio:

```
typedef int Lunghezza;
```

- crea un sinonimo di int:

```
Lunghezza l1,l2;
```

```
Lunghezza l3[10];
```

Definire nuovi tipi

Questa possibilità è utile soprattutto per le strutture infatti definendo come nuovo tipo di dato una struttura si semplifica la dichiarazione delle variabili di tipo struttura

Esempio:

```
struct libro{
```

```
    char titolo[100];
```

```
    char autore[50];
```

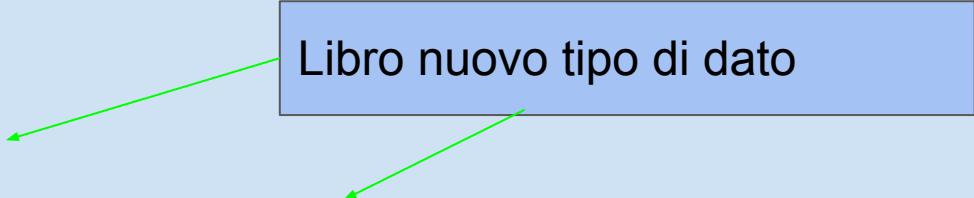
```
};
```

```
typedef struct libro Libro;
```

da adesso posso scrivere **Libro**

invece di **struct libro**

Libro nuovo tipo di dato



Libro l1;