




**UNIVERSITÀ DEGLI STUDI
DEL SANNIO** Benevento
DING
DIPARTIMENTO DI INGEGNERIA

CORSO DI "PROGRAMMAZIONE I"

Prof. Franco FRATTOLILLO
Dipartimento di Ingegneria
Università degli Studi del Sannio


Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 1



04104122 Le strutture

- **Esistono due grandi categorie di dati in C:**
 - **Dati non strutturati** *semplici*
 - int, char, float, double, pointer
 - **Dati strutturati**
 - **Omogenei** *collezioni di var. dello stesso tipo sotto un solo nome logico*
 - array
 - **Eterogenei**
 - struct
- **Una struttura è una collezione di una o più variabili (membri o campi della struttura), normalmente di tipi diversi, raggruppate sotto uno stesso nome**


Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 2



Perché usare le strutture

- **Quando** in un programma **si vogliono rappresentare dati appartenenti al mondo reale**, i tipi di dati semplici, quali interi, caratteri, ecc. **non sono sufficienti**
 - **spesso ci si deve riferire ad oggetti che sono aggregati di dati diversi**
- • ad esempio, un libro è un'entità composta da un titolo, un autore, un editore, l'anno di pubblicazione, il testo
- Le strutture **sono utili anche**
 - • per definire record da memorizzare in file
 - • combinate con i puntatori, per creare strutture dati dinamiche (liste, alberi, grafi ...)

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 3



Esempi di strutture

- **Per dichiarare una struttura si usa la parola riservata struct**
- **La definizione di una struttura crea un nuovo tipo**

→ • dipendente è il nome della struttura

→ • È un nuovo tipo che può essere usato per dichiarare variabili di tipo dipendente

→ • Ogni variabile di tipo dipendente consta di 3 campi, l'intero codice, l'array di caratteri nome e il float stipendio

```
struct card {
    char *face;
    char *suit;
};
```

```
struct dipendente {
    int codice;
    char nome[30];
    float stipendio;
};
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 4

FRANCO
LILLO

DIPENDENTE:

int	codice
char	nome
float	stipendio

La dichiarazione di variabili struct

- Creato il tipo, si possono dichiarare le variabili

```
struct dipendente programmatore;
struct card oneCard, deck[52], *cPtr;
```

- Oppure, è possibile fondere creazione del tipo e dichiarazione di variabili

```
struct card {
    char *face;
    char *suit;
} oneCard, deck[52], *cPtr;
```

In questo caso, il tag può essere omissso

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 5

L'inizializzazione delle strutture

- • Le strutture possono essere inizializzate
- • Esempi:

```
struct dipendente programmatore = {487, "Marco", 12000.35};
```

```
struct dipendente {
    int codice;
    char nome[30];
    float stipendio;
} programmatore = {487, "Marco", 12000.35};
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 6

L'assegnazione delle strutture

- Le strutture possono essere globalmente assegnate

i vettori no => puntatori costanti alle feste

- Esempio:

```
struct dipendente prog1 = {487, "Marco", 12000.35};
struct dipendente prog2;
...
prog2 = prog1;
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 7

La selezione dei campi

- Per accedere al singolo campo di una struttura bisogna usare la cosiddetta "dot notation"

⇒ <nome variabile struttura>.<nome campo>

- Esempi:

```
programmatore.codice = 487;
strcpy(programmatore.nome, "Marco");
programmatore.stipendio = 12000.35;
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 8

Altre operazioni sulle strutture

- Altre operazioni che possono essere effettuate su strutture sono:
 - ottenere l'indirizzo di una struttura (con &)
 - usare l'operatore `sizeof` per determinare l'ingombro di una struttura in termini di byte

Strutture ed array

- La dichiarazione di un array di strutture si realizza così:

```
struct matita {
    int durezza;
    char *fabbricante;
    int numero;
};
```

```
struct matita {
    int durezza;
    char *fabbricante;
    int numero;
} m[3];
```

→ puntatore a carattere

```
main() {
    struct matita m[3];
    m[0].durezza = 2;
    m[2].numero = 400;
    m[1].fabbricante = "Modini";
}
```

m = [] = [] = []

Esempio

- Gli array di strutture sono utili al posto dei vettori paralleli o bidimensionali
 - ad esempio, un programma che conta le occorrenze delle parole chiavi del C, invece di essere organizzato con due vettori paralleli
- | | | | | |
|--------|------|-------|------|-------|
| char * | auto | break | | while |
| int | 3 | 5 | ... | 1 |
- può essere organizzato con un vettore di strutture
 - ogni elemento dell'array è una struttura a due campi che specifica una parola chiave ed il relativo numero di occorrenze

Esempio

```
struct key {
    char *word;
    int count;
} keytab[NKEY] = {"auto", 0,
                  "break", 0,
                  ...
                  "while", 0};

...
for(i = 0; i < NKEY; i++)
    printf("%s è presente %d volte\n", keytab[i].word,
           keytab[i].count);
```

token → scanf
linee → gets

token ← char []
occ. ← int

```
struct Token {
    char str[30];
    int occ;
};
```

Token Test0 [100];

while	piippo		
3	4		

Strutture innestate

- Una struttura può contenere anche altre strutture

```
struct progetto {
    int mesi;
    double budget;
    char argomento[100];
    struct dipendente pgm[100];
} prgSoft;
```

- L'inserimento del 15° programmatore nel progetto può avvenire così:
- ```
strcpy(prgSoft.pgm[14].nome, "Marco");
```

## Esempio

- Rappresentiamo mediante una struttura l'oggetto rettangolo

→ In una rappresentazione cartesiana, un rettangolo è univocamente determinato dai suoi vertici in alto a destra ed in basso a sinistra

```
struct point {
 int x;
 int y;
};
struct rectangle {
 struct point pt1;
 struct point pt2;
};
```

## Un'osservazione

- • Una struttura non può contenere un'istanza di se stessa
- • Una struttura può però contenere un puntatore ad una struttura dello stesso tipo
  - È questa la chiave per realizzare strutture dati dinamiche

## Un esempio di puntatore a struttura

```
struct matita {
 int durezza;
 char fornitore[40];
} m;
...
struct matita (*matPunt) = &m;
(*matPunt).durezza = 3;
strcpy((*matPunt).fornitore, "Mondini");
...
```

puntatore e strutture

Le parentesi sono importanti in quanto la precedenza dell'operatore . è superiore a quella di \*

### L'operatore ->

- Per semplificare l'accesso ai membri di una struttura, quando si usano i puntatori, è disponibile l'operatore -> *freccia*
- <puntatore struttura> -> <membro struttura>
- nell'esempio precedente, possiamo scrivere
 

```
matPunt->durezza = 3;
strcpy(matPunt->fornitore, "Mondini");
```
- nel caso di strutture innestate, l'operatore -> può essere utilizzato ripetutamente (è associativo a sinistra)
 

```
struct cartella {
 ...
 struct matita *m;
} *q;
...
... q->m->fornitore ...
... q->m->durezza ...
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 17

### Il passaggio delle strutture

- Le strutture sono passate alle funzioni **per valore**
  - pertanto, la funzione opera su una copia locale della struttura
- È possibile il passaggio **per riferimento**, usando come parametro un puntatore alla struttura
  - in tal modo, si consente alla funzione di modificare la struttura

*array solo riferimento*

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 18

### L'aritmetica dei puntatori a strutture

- Sui puntatori a strutture è possibile applicare un'aritmetica, così come per gli altri puntatori
  - occorre soltanto **tenere presente che gli operatori d'accesso . e -> hanno la precedenza massima e che sono operatori associativi a sinistra**

```
struct Str{
 int len;
 char *str;
} *p;
```

```
struct Str s;
s.len=2;
s.str="pippo";
p=&s; p punta a s
```

```
... ++p->len ...
/* è uguale a */
... ++(p->len) ...
/* che è diverso da */
... (++p)->len ...
... ++p(->len) ... /* SBAGLIATO */
```

Per risolvere il problema usare le parentesi

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 19

### La dichiarazione typedef

- typedef consente di dichiarare nuovi tipi sulla base di tipi esistenti**
- Ad esempio:
 

```
typedef char* Stringa
```
- definisce Stringa come un alias del tipo char \***
- Pertanto, si può scrivere:
 

```
Stringa str1, str2;
```
- Esempio:
 

```
typedef struct punto {...} PIXEL;
PIXEL center, *origin;
/* invece di struct punto center, *origin; */
```

Franco FRATTOLILLO - Dipartimento di Ingegneria - Università degli Studi del Sannio Corso di "Programmazione I" - Corso di Laurea in Ingegneria Informatica / ExAT 20



## La dichiarazione typedef

- L'uso di typedef consente di aumentare la chiarezza e la leggibilità del programma
  - forse ...
- L'uso di typedef consente anche di elevare la portabilità del programma
  - dovendo portare su un altro sistema un programma che ha tipi dipendenti dall'architettura, è sufficiente modificare le typedef senza che vi sia la necessità di modificare le dichiarazioni di variabili