

PROGRAMMAZIONE 2: SPERIMENTAZIONI

Lezione 3 – Strutture in C



Agenda

- Introduzione
- Definizione di strutture
- Strutture autoreferenziali
- Definizione di variabili di tipo struttura
- Operazioni eseguibili sulle strutture
- Accesso ai membri delle strutture
- Strutture e funzioni
- Usare la parola chiave `typedef`



Video lezione disponibile su YouTube: <https://youtu.be/e43EQMjGFYY>

Introduzione

- Le **strutture**, talvolta chiamate **aggregati**, sono collezioni di variabili collegate sotto un unico nome.
- Le strutture possono contenere **variabili di tipi differenti** (contrariamente agli array che contengono solo elementi dello stesso tipo di dati).
- I puntatori e le strutture facilitano la formazione di strutture di dati più complesse come **liste collegate**, code, pile e alberi.

Definizione di strutture

- Le strutture sono **tipi di dati derivati** in quanto costruite usando *oggetti* di altri tipi.
 - In C le strutture sono definite tramite la parola chiave **struct** e un'**etichetta**.
 - L'etichetta, insieme alla parola chiave **struct**, **sarà utile per definire le variabili di tipo struttura**.
 - Le variabili dichiarate entro le parentesi graffe della struttura sono dette i **membri** della struttura (all'interno della stessa struttura non possono esserci membri con lo stesso nome).
 - Ogni definizione di struttura **deve** terminare con un punto e virgola.

Definizione di strutture

- Esempio di struttura

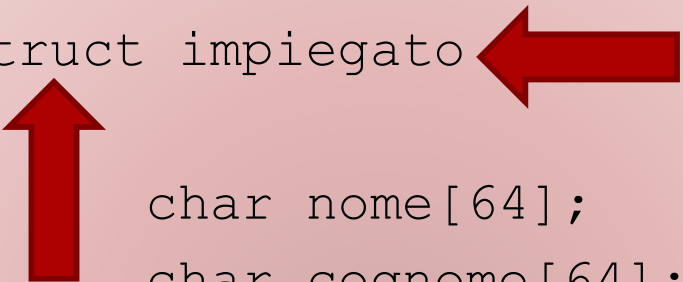
```
#include <stdio.h>

struct impiegato
{
    char nome[64];
    char cognome[64];
    unsigned int eta;
    char sesso;
    double stipendio;
};
```

Definizione di strutture

```
#include <stdio.h>

struct impiegato
{
    char nome[64];
    char cognome[64];
    unsigned int eta;
    char sesso;
    double stipendio;
};
```

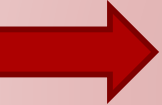


- La parola chiave **struct** *introduce una definizione di struttura*.
- L'identificatore **impiegato** è l'**etichetta** della struttura.

Definizione di strutture

```
#include <stdio.h>

struct impiegato
{
    char nome[64];
    char cognome[64];
    unsigned int eta;
    char sesso;
    double stipendio;
};
```




- Le variabili dichiarate entro le parentesi della struttura (nome, cognome, età, ecc...) sono dette **membri**.
- I membri di una struttura possono essere variabili dei tipi primitivi (int, float, ecc...) o aggregati come array o altre strutture.
- I membri di una struttura possono essere di tipi differenti.

Definizione di strutture

```
#include <stdio.h>

struct impiegato
{
    char nome[64];
    char cognome[64];
    unsigned int eta;
    char sesso;
    double stipendio;
};
```



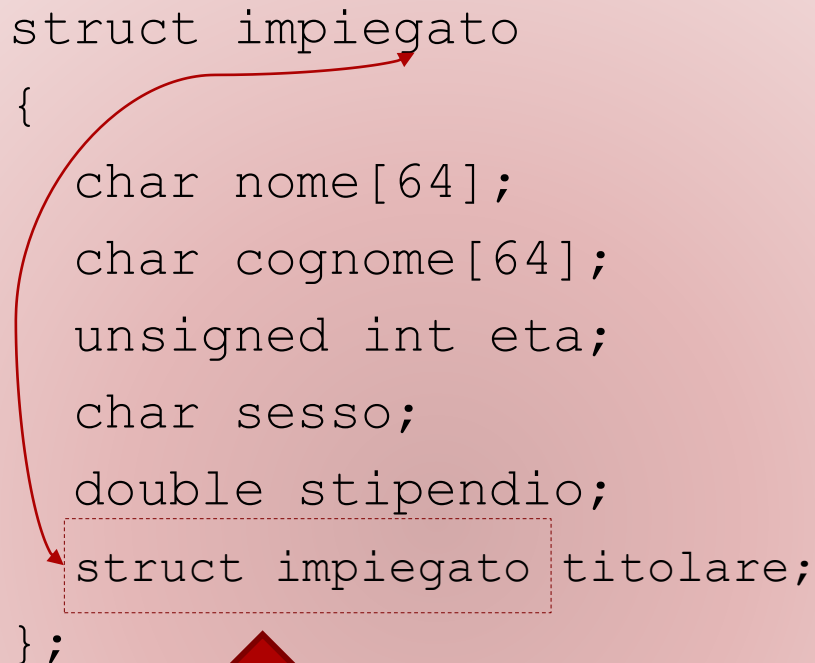
- Ogni definizione di struttura **deve terminare con un punto e virgola.**
- *Dimenticare il punto e virgola al termine della definizione di una struttura costituisce un errore di sintassi, che impedisce la compilazione del programma.*

Strutture autoreferenziali

- Come notato nell'esempio precedente, i membri di una struttura possono essere vari.
- Una variabile di tipo `struct` però **non può essere dichiarata nella definizione di quello stesso tipo di `struct`.**

Strutture autoreferenziali

```
struct impiegato
{
    char nome[64];
    char cognome[64];
    unsigned int eta;
    char sesso;
    double stipendio;
    struct impiegato titolare;
};
```



- In questa definizione si sta cercando di definire un membro titolare che a sua volta è un impiegato.
- Una variabile di tipo struct **non può essere dichiarata nella definizione di quello stesso tipo di struct.**
- La definizione della struttura impiegato è quindi **errata.**

Strutture autoreferenziali

- Per ovviare al problema di cui sopra è possibile inserire un **puntatore** a quel tipo di `struct`.
- **Una struttura contenente un membro che è un puntatore allo stesso tipo di struttura è detta struttura autoreferenziale.**

Le strutture autoreferenziali sono usate per costruire strutture dati collegate (liste, pile, code).

Definizione di variabili di tipo struttura

- Le definizioni di strutture **non** riservano alcuno spazio in memoria, ma **creano un nuovo tipo di dato utilizzabile per definire le variabili**.
- Le variabili di tipo struttura sono definite come le altre variabili.
- Ad esempio:

```
struct impiegato imp;  
// dichiara una variabile imp di tipo impiegato  
  
struct impiegato vimp[20];  
// dichiara un vettore contenente 20 elementi (da 0 a 19) di tipo impiegato  
  
struct impiegato *impPtr;  
// dichiara un puntatore a variabili di tipo impiegato
```

Operazioni eseguibili sulle strutture

- Le operazioni eseguibili sulle strutture sono:
 - 1) assegnare variabili `struct` a variabili `struct` dello stesso tipo;
 - 2) accedere all'indirizzo di memoria (tramite `&`) di una variabile di tipo struttura;
 - 3) accedere ai membri di una variabile di tipo struttura;
 - 4) utilizzare l'operatore `sizeof` per determinare la dimensione di variabili di tipo `struct`.

Operazioni eseguibili sulle strutture

- Le strutture **non** possono essere confrontate tra di loro usando gli operatori `==` e/o `!=` poiché la dimensione dei dati di un particolare tipo e le caratteristiche di salvataggio in memoria centrale sono dipendenti dalla macchina; lo è anche la rappresentazione dei dati di una struttura stessa.
- **L'esecuzione di un confronto tra strutture comporta un errore in fase di compilazione.**

Inizializzazione di strutture

- Le variabili di tipo struttura possono essere inizializzate usando liste di inizializzatori come con gli array.

```
struct impiegato imp1 = {"Andrea", "Rossi", 30, 'M', 30000};  
struct impiegato imp2 = {"Mario", "Bianchi"};
```

- I membri non inizializzati sono impostati a 0 o NULL in base al loro tipo.
- Le variabili di tipo struttura possono anche essere inizializzate a valori di un'altra struttura dello stesso tipo (in automatico tutti i membri assumono gli stessi valori).

```
struct impiegato imp3 = imp1;
```

Inizializzazione di strutture

es1.c

```

1  #include <stdio.h>
2
3  struct impiegato
4  {
5      char nome[64];
6      char cognome[64];
7      unsigned int eta;
8      char sesso;
9      double stipendio;
10 };
11
12
13 int main(void)
14 {
15     struct impiegato imp1 = {"Andrea", "Rossi", 30, 'M', 30000};
16     struct impiegato imp2 = {"Mario", "Bianchi"};
17     struct impiegato imp3 = imp1;
18
19     printf("Impiegato 1\n");
20     printf("Nome imp1: %s\n", imp1.nome);
21     printf("Cognome imp1: %s\n", imp1.cognome);
22     printf("Eta imp1: %d\n", imp1.eta);
23     printf("Sesso imp1: %c\n", imp1.sesso);
24     printf("Stipendio imp1: %5.2f\n", imp1.stipendio);
25     printf("-----\n");
26     printf("Impiegato 2\n");
27     printf("Eta imp2: %d\n", imp2.eta);
28     printf("Sesso imp2: %c\n", imp2.sesso);
29     printf("Stipendio imp2: %5.2f\n", imp2.stipendio);
30     printf("-----\n");
31     printf("Impiegato 3\n");
32     printf("Eta imp3: %d\n", imp3.eta);
33     printf("Sesso imp3: %c\n", imp3.sesso);
34     printf("Stipendio imp3: %5.2f\n", imp3.stipendio);
35
36     /* ERRORE in fase di compilazione!
37     Le strutture non sono paragonabili direttamente
38     if (imp1==imp3)
39     {
40         printf("Le strutture sono uguali.");
41     }
42     */
43
44     return 1;
45 }

```

```

C:\programmazione_2\lezione_2_strutture>a.exe
Impiegato 1
Nome imp1: Andrea
Cognome imp1: Rossi
Eta imp1: 30
Sesso imp1: M
Stipendio imp1: 30000.00
-----
Impiegato 2
Eta imp2: 0
Sesso imp2:
Stipendio imp2: 0.00
-----
Impiegato 3
Eta imp3: 30
Sesso imp3: M
Stipendio imp3: 30000.00
C:\programmazione_2\lezione_2_strutture>

```


Accesso ai membri delle strutture

- È possibile accedere ai membri delle strutture in **due** modi:
 - 1) tramite l'**operatore di membro di struttura**, il punto .
 - 2) tramite l'**operatore di puntatore a struttura** ->
(senza spazi tra - e >)

```
struct impiegato imp = {"Andrea", "Rossi", 30, 'M', 30000};  
struct impiegato *impPtr;  
  
printf("Nome impiegato: %s\n", imp.nome);  
impPtr = &imp;  
// fa puntare impPtr a imp  
printf("Nome impiegato: %s\n", impPtr->nome);  
// impPtr->nome equivale a (*impPtr).nome
```

Accesso ai membri delle strutture

```

1  #include <stdio.h>
2  #include <string.h> // per la strcpy
3
4  struct impiegato
5  {
6      char nome[64];
7      char cognome[64];
8      unsigned int eta;
9      char sesso;
10     double stipendio;
11 };
12
13
14 int main(void)
15 {
16     struct impiegato imp = {"Andrea","Rossi", 30, 'M', 30000};
17     struct impiegato vimp[20];
18     struct impiegato *impPtr;
19
20     printf("Nome impiegato: %s\n",imp.nome);
21     impPtr = &imp; // il puntatore punta alla variabile imp
22     printf("Nome impiegato: %s\n",impPtr->nome);
23
24     vimp[0].eta=50;
25     printf("Eta impiegato: %d\n",vimp[0].eta);
26
27     impPtr = vimp; // il puntatore punta alla prima cella di vimp
28     // equivale a impPtr=&v[0]
29     impPtr->eta=60;
30     printf("Eta impiegato [0]: %d\n",vimp[0].eta);
31     ++impPtr; // sposta il puntatore di una cella quindi passa a [1]
32     // equivale a impPtr = impPtr + 1;
33     impPtr->eta=30;
34     printf("Eta impiegato [1]: %d\n",vimp[1].eta); // oppure impPtr->eta
35     // impPtr->nome = "Mario"; ERRORE!
36     strcpy(impPtr->nome,"Mario"); // copia "Mario" nel membro nome
37     printf("Nome impiegato [1]: %s\n",vimp[1].nome); // impPtr->nome
38
39     return 1;
40 }

```

es2.c

```

C:\programmazione_2\lezione_2_strutture>a.exe
Nome impiegato: Andrea
Nome impiegato: Andrea
Eta impiegato: 50
Eta impiegato [0]: 60
Eta impiegato [1]: 30
Nome impiegato [1]: Mario
C:\programmazione_2\lezione_2_strutture>

```

Strutture e funzioni

- Le strutture possono essere passate alle funzioni nei seguenti modi:
 - 1) utilizzando i loro membri individuali;
 - 2) utilizzando l'intera struttura;
 - 3) utilizzando un puntatore alla struttura.

Strutture e funzioni

- Quando le strutture o i loro membri individuali sono passati a una funzione, lo sono per **valore** (quindi i membri di una struttura non possono essere modificati).
- Per passare a una funzione una struttura per riferimento è necessario passare l'indirizzo di una variabile di tipo struttura.
- Gli **array di strutture**, *come tutti gli altri array*, sono automaticamente passati per **riferimento**.
- Se una struttura è passata per valore, gli array al suo interno sono passati per valore quindi non rimane traccia delle modifiche agli stessi a conclusione della funzione.



Passare le strutture per riferimento è più efficiente che passare le strutture per valore (che richiede la copiatura dell'intera struttura).

Strutture e funzioni

```

1  #include <stdio.h>
2
3  struct impiegato
4  {
5      char nome[64];
6      char cognome[64];
7      unsigned int eta;
8      char sesso;
9      double stipendio;
10 };
11
12 void aumenta_eta_v1(struct impiegato aImp);
13 void aumenta_eta_v2(struct impiegato *aImp);
14
15 int main(void)
16 {
17     struct impiegato imp = {"Andrea", "Rossi", 30, 'M', 30000};
18
19     printf("Eta dell'impiegato PRIMA della funzione _v1: %d\n", imp.eta);
20     aumenta_eta_v1(imp);
21     printf("Eta dell'impiegato DOPO la funzione _v1: %d\n", imp.eta);
22     printf("-----\n");
23     printf("Eta dell'impiegato PRIMA della funzione _v2: %d\n", imp.eta);
24     aumenta_eta_v2(&imp);
25     printf("Eta dell'impiegato DOPO la funzione _v2: %d\n", imp.eta);
26     return 1;
27 }
28
29 void aumenta_eta_v1(struct impiegato aImp) // aImp è per valore
30 {
31     aImp.eta++;
32     printf("Eta dell'impiegato NELLA funzione _v1: %d\n", aImp.eta);
33 }
34
35 void aumenta_eta_v2(struct impiegato *aImp) // aImp è per riferimento
36 {
37     aImp->eta++;
38     printf("Eta dell'impiegato NELLA funzione _v2: %d\n", aImp->eta);
39 }
    
```

es3.c

```

C:\programmazione_2\lezione_2_strutture>a.exe
Eta dell'impiegato PRIMA della funzione _v1: 30
Eta dell'impiegato NELLA funzione _v1: 31
Eta dell'impiegato DOPO la funzione _v1: 30
-----
Eta dell'impiegato PRIMA della funzione _v2: 30
Eta dell'impiegato NELLA funzione _v2: 31
Eta dell'impiegato DOPO la funzione _v2: 31
C:\programmazione_2\lezione_2_strutture>_
    
```

Usare la parola chiave `typedef`

- La parola chiave `typedef` fornisce un meccanismo per creare sinonimi (o *alias*) per tipi di dati precedentemente definiti.
- I nomi per i tipi di strutture sono spesso definiti con `typedef` per creare nomi di tipo più brevi.
- Si noti che `typedef` non crea un nuovo tipo, bensì un nuovo nome di tipo che può essere utilizzato come alias di un nome esistente.

Usare la parola chiave `typedef`

```
typedef struct
{
    char nome[64];
    char cognome[64];
    unsigned int eta;
    char sesso;
    double stipendio;
} Impiegato;
```

```
Impiegato imp;
```



equivale a

```
struct impiegato imp;
```

Usare la parola chiave `typedef`



Scrivere in maiuscolo la prima lettera dei nomi creati con `typedef` per evidenziare che sono sinonimi (alias).



Usare `typedef` può contribuire a rendere un programma più portabile.



Usare `typedef` può contribuire a rendere un programma più leggibile e mantenibile.

Usare la parola chiave typedef



```

1  #include <stdio.h>
2
3  typedef struct
4  {
5      char nome[64];
6      char cognome[64];
7      unsigned int eta;
8      char sesso;
9      double stipendio;
10 } Impiegato;
11
12 void aumenta_eta_v1(Impiegato aImp);
13 void aumenta_eta_v2(Impiegato *aImp);
14
15 int main(void)
16 {
17     Impiegato imp = {"Andrea", "Rossi", 30, 'M', 30000};
18
19     printf("Eta dell'impiegato PRIMA della funzione _v1: %d\n", imp.eta);
20     aumenta_eta_v1(imp);
21     printf("Eta dell'impiegato DOPO la funzione _v1: %d\n", imp.eta);
22     printf("-----\n");
23     printf("Eta dell'impiegato PRIMA della funzione _v2: %d\n", imp.eta);
24     aumenta_eta_v2(&imp);
25     printf("Eta dell'impiegato DOPO la funzione _v2: %d\n", imp.eta);
26     return 1;
27 }
28
29 void aumenta_eta_v1(Impiegato aImp) // aImp è per valore
30 {
31     aImp.eta++;
32     printf("Eta dell'impiegato NELLA funzione _v1: %d\n", aImp.eta);
33 }
34
35 void aumenta_eta_v2(Impiegato *aImp) // aImp è per riferimento
36 {
37     aImp->eta++;
38     printf("Eta dell'impiegato NELLA funzione _v2: %d\n", aImp->eta);
39 }
    
```

es4.c

FINE PRESENTAZIONE

