Strutture

Prof. Orazio Mirabella

Strutture (tipo struct)

Una **struttura** (o **record**) serve per **aggregare** elementi (anche di tipo diverso) sotto un unico nome.

Sono un tipo di dato **derivato**, ovvero costruito a partire da dati di altri tipi.

Definizione di una struttura

Esempio:
```
struct data {
        int giorno;
        int mese;
        int anno;
        };
```

oppure

```
struct data {
int giorno;
char *mese;
int anno;
};
```

- parola chiave `struct` introduce la definizione della struttura
- `data` `e l'**etichetta** della struttura, che attribuisce un nome alla definizione della struttura
- `giorno`, `mese`, `anno` sono i **campi** (o **membri**) della struttura

Strutture

- In generale una struttura è definita dalla sintassi:

  struct *nome_struttura* {

  tipo_membro nome_membro1;

  tipo_membro nome_membro2;

  ...

  tipo_membro nome_membroN;

  };

Campi di una struttura

- devono avere nomi univoci all'interno di una struttura
- strutture diverse possono avere campi con lo stesso nome
- i nomi dei campi possono coincidere con nomi di variabili o funzioni

Esempio:

```
int x;
struct a { char x; int y; };
struct b { int w; float x; };
```

- possono essere di tipo diverso (semplice o altre strutture)
- un campo di una struttura non può essere del tipo struttura che si sta definendo

Esempio:

```
struct s { int a; struct s next; };
```

- un campo può però essere di tipo puntatore alla struttura

Esempio:

```
struct s { int a;
struct s *next; };
```

- la definizione della struttura non provoca allocazione di memoria, ma introduce un nuovo tipo di dato

Dichiarazione di variabili di tipo struttura

- Una volta definita una struttura, *nome_struttura* diviene un nuovo tipo a tutti gli effetti. Si possono allora definire variabili il cui tipo è *nome_struttura*:
- struct *nome_struttura nome_variabile*;

Esempio: struct data oggi, appelli[10], *pd;

- oggi è una variabile di tipo struct data
- appelli `e un vettore di 10 elementi di tipo struct data
- pd `e un puntatore a una struct data

Una variabile di tipo struttura può essere dichiarata contestualmente alla definizione della struttura.

Esempio:
```
struct studente {
char nome[20];
long matricola;
struct data ddn;
} s1, s2;
```

In questo caso si può anche **omettere l'etichetta struct** davanti alle variabili.

Dichiarazione di variabili di tipo struttura

```
struct automobile {
char *marca;
char *modello;
int venduto;
};
struct automobile a1, a2;
```

Oppure

```
struct automobile {
char *marca;
char *modello;
int venduto;
} a1, a2;
```

Operazioni sulle strutture

- Si possono assegnare variabili di tipo struttura a variabili **dello stesso tipo** struttura.
- *Esempio:* struct data d1, d2;
- ...
- d1 = d2;
- Nota: questo permette di assegnare interi vettori.
- *Esempio:* struct matrice { int elementi[10][10]; };
- struct matrice a, b;
- ...
- a = b;
- **Non** `e possibile effettuare il confronto tra due variabili di tipo struttura.
- *Esempio:* struct data d1, d2;
- if (d1 == d2) ... **Errore!**
- Motivo: una struttura può contenere dei "**buchi**" dovuti alla necessità di allineare i campi con le parole di memoria.

Equivalenza fra strutture

- L'equivalenza di tipo tra strutture è **per nome**.
- *Esempio:* struct s1 { int i; };
- struct s2 { int i; };
- struct s1 a, b;
- struct s2 c;
- a = b; **OK**
- a = c; **Errore**, perchè a e c non sono dello stesso tipo.

- Si può ottenere l'indirizzo di una variabile di tipo struttura tramite l'operatore &.

- Si può rilevare la dimensione di una struttura con sizeof.
- *Esempio:* sizeof(struct data) restituisce la dimensione del tipo (struct data), ossia il numero di unità di memoria che verrebbero occupate da un oggetto qualsiasi di tipo struct data.

- Attenzione: **non è** detto che la dimensione di una struttura sia pari alla somma delle dimensioni dei singoli campi (ci possono essere **buchi**).

Accesso ai campi della struttura

- Avviene tramite l'**operatore punto**
- *Esempio:* struct data oggi;
- oggi.giorno = 8;
- oggi.mese = 5;
- oggi.anno = 2002;
- printf("%d %d %d", oggi.giorno, oggi.mese, oggi.anno);
- La sintassi generale con cui si fa riferimento a un membro è:

  nome_variabile_struttura.nome_membro

/* Esempio di definizione di una struttura */

```
#include <stdio.h>
struct automobile {
char *marca;
char *modello;
int venduto;
};

main()
{
struct automobile a1, a2;
a1.marca = "FERRARI";
a1.modello = "F40";
a1.venduto = 200;
a2.marca = "OPEL";
a2.modello = "ASTRA";
a2.venduto = 1200;
printf("marca auto = %s\n", a1.marca);
printf("modello auto = %s\n", a1.modello);
printf("vendute = %d\n", a1.venduto);
printf("marca auto = %s\n", a2.marca);
printf("modello auto = %s\n", a2.modello);
printf("vendute = %d\n", a2.venduto);
}
```

L'esecuzione del programma produrrà il seguente risultato:
marca auto = FERRARI
modello auto = F40
vendute = 200
marca auto = OPEL
modello auto = ASTRA
vendute = 1200

Accesso alla struttura tramite un puntatore.

- *È possibile fare riferimento a una variabile struttura anche attraverso un puntatore, contenente l'indirizzo della variabile di tipo struttura:*
- *Es.1: struct data *pd, oggi, compleanno;*

```
pd = &oggi;
(*pd).giorno = 31;
(*pd).mese = "Gennaio";
(*pd).anno = 2001;
```

- *Attraverso il puntatore pd si possono raggiungere i membri della variabile struttura oggi.*
- *Es2:* struct data *pd;

```
pd = malloc(sizeof(struct data));
(*pd).giorno = 8;
(*pd).mese = 5;
(*pd).anno = 2002;
```

- N.B. Ci vogliono le () perchè "." ha priorità più alta di "*".

Operatore freccia

- *L'operatore freccia -> permette di accedere direttamente ai membri di una variabile strutturata puntata da un puntatore. :*

- *struct data *pd, oggi, compleanno;*
- *…..*
- *pd = &oggi;*
- *pd->giorno = 31*
- *pd->mese = "Gennaio";*
- *pd->anno = 2001;*

- I puntatori a struttura sono molto usati specialmente per passare una struttura a una funzione, e per far sì che una funzione ritorni un risultato di tipo struttura

Passaggio di una struttura a una funzione

```
int numero_mese(struct data *dt)
{
if(dt->mese == "Gennaio")
return(1);
else
if(dt->mese == "Febbraio")
return(2);
else
...
if(dt->mese == "Dicembre")
return(12)
else
return(0);
}
```

Alla funzione numero_mese viene passato un puntatore a variabile di tipo data e si ottiene il numero del mese relativo alla data puntata dal puntatore. Nel caso in cui il nome del mese non corrisponda ad alcuno di quelli conosciuti, la funzione ritorna un codice di errore pari a 0.

Inizializzazione di struct

- Si può usare una sintassi compatta analoga a quella usata per inizializzare gli array:
-
```
struct automobile {
char *marca;
char *modello;
int venduto;
};
struct automobile a = {"FERRARI", "F40", 200};
```

Oppure

-
```
struct data {
int giorno;
char *mese;
int anno;
} oggi = {25, "Dicembre", 2001};
```

- Questo tipo di inizializzazione è ammesso solo se le corrispondenti variabili sono di tipo globale, cioè se sono definite all'esterno di un blocco

Esercizio 1 sulle strutture

- **Si scriva un programma in linguaggio C che tramite:**
- **la funzione leggi(), legga a terminale i dati di N studenti (con N definito** come la costante 4) costituiti da Nome, Cognome, Voto, e li inserisca in un vettore;
- calcoli con la funzione media() la media dei voti;
- stampi a terminale i nominativi di ciascuno studente;
- stampi poi a terminale il voto medio.
- Ad esempio, avendo in ingresso:
  Alberani Luigi 30
- Vettori Piero 33
- Zanetti Lorenzo18
- stampi:
  Alberani Luigi Vettori Piero Zanetti Lorenzo
- Voto medio: 27.0

Esercizio 1 sulle strutture

```
#include <stdio.h>
#define N 4

struct studente{
char Nome[20];
char Cognome[20];
int Voto;};

void leggi(int n, struct studente Vet[]);
float media(int n, struct studente Vet[]);

main(){
int i;
struct studente V[N];
leggi(N,V);
for(i=0;i<N;i++){
printf("%s\n",V[i].Nome);
printf("%s\n", V[i].Cognome);
}
printf("\nVoto medio: %f\n",media(N,V));
}

void leggi(int n, struct studente Vet[]){
int i,j;
for(i=0;i<n;i++){
printf("Inserisci Nome, Cognome e Voto");
scanf("%s",Vet[i].Nome);
scanf("%s",Vet[i].Cognome);
scanf("%d",&Vet[i].Voto);
}
}
float media(int n, struct studente Vet[]){
int i;
float m=0.0;
for(i=0;i<n;i++)
m=m+Vet[i].Voto;
return m/n;
}
```