

Tipi strutturati in C

Strutture

Strutture

- Una *struttura* è una **collezione finita di variabili non necessariamente dello stesso tipo**, ognuna identificata da un *nome*
- Una **struttura** (o **record**) serve quindi per **aggregare** elementi (anche di tipo diverso) sotto un unico nome.

Sintassi

```
struct <etichetta> {  
    <definizione-di-variabili>  
};
```

Definizione di strutture

Esempio di definizione di struttura:

```
struct persona {  
    char nome[20];  
    int età;  
    float stipendio;  
};
```

Con l'etichetta si attribuisce un nome alla definizione della struttura

nome, età e stipendio sono i **campi** (o **membri**) della struttura

- **ATTENZIONE:** La definizione della struttura non provoca allocazione di memoria, ma introduce un nuovo tipo di dato

Dichiarazione di variabili struttura

Esempio:

```
struct persona p1, p2;  
struct persona elenco[50];
```

p1 e p2 sono variabili di tipo *struct persona*

elenco è un array di 50 elementi di tipo *struct persona*

- Una variabile di tipo struttura può essere dichiarata **contestualmente** alla definizione della struttura.
- In questo caso si può anche **omettere l'etichetta** di struttura.

```
struct data {  
    int giorno;  
    int mese;  
    int anno;  
} d1,d2;
```

d1 e d2 sono variabili di tipo *struct data*

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

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

Esempi
corretti

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

ERRORE

Accesso ai campi di una struttura

- Una volta definita una variabile struttura, *si accede ai singoli campi* mediante la **notazione puntata**

```
struct punto {
    int x, y;
} p1, p2 ;
...
p1.x = 10;
p1.y = -2;
p2.x = 5;
p2.y = 7;
```

```
struct data {
    int giorno, mese, anno;
} d1, d2 ;
...
d1.giorno = 25;
d1.mese = 12;
d1.anno = 2003;
printf("%d-%d-%d", d1.giorno,
        d1.mese, d1.anno);
```

- Ogni campo si usa come una normale variabile del tipo corrispondente a quello del campo

Operazioni sulle strutture

- Si possono assegnare variabili di tipo struttura a variabili **dello stesso tipo** struttura.

Esempio:

```
struct data d1, d2;  
...  
d1 = d2;
```

- **Non è** possibile effettuare il confronto tra due variabili di tipo struttura.

Esempio:

```
struct data d1, d2;  
if (d1 == d2) ... Errore!
```

Esempio

- **PROBLEMA:** leggere le coordinate di un punto in un piano e modificarle a seconda dell'operazione richiesta
 1. proiezione sull'asse X
 2. proiezione sull'asse Y
 3. traslazione di DX e DY
- **Specifica:**
 - leggere le coordinate di input e memorizzarle in una struttura
 - leggere l'operazione richiesta
 - effettuare l'operazione
 - stampare il risultato

Esempio

```
#include <stdio.h>
main() {
    struct punto{float x,y;} P;
    unsigned int op;
    float Dx, Dy;
    printf("ascissa? "); scanf("%f",&P.x);
    printf("ordinata? "); scanf("%f",&P.y);
    printf("%s\n","operazione(0,1,2,3)?"); scanf("%d",&op);
    switch (op) {
        case 1: P.y=0;break;
        case 2: P.x=0; break;
        case 3: printf("%s","Traslazione?");
                scanf("%f%f",&Dx,&Dy);
                P.x=P.x + Dx;
                P.y=P.y + Dy;
                break;
        default: ;
    }
    printf("%s\n","Le nuove coordinate sono ");
    printf("%f %f\n",P.x,P.y);
}
```