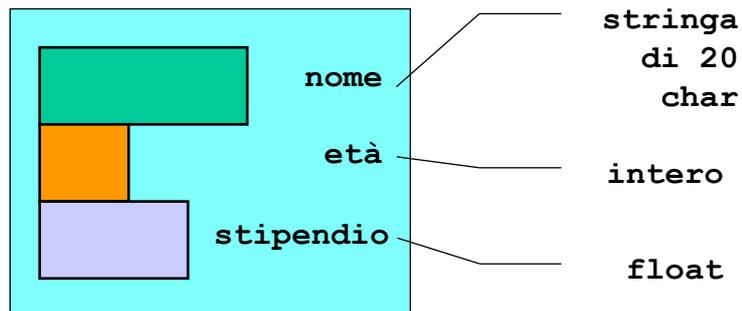


STRUTTURE

- Una *struttura* è una collezione finita di variabili non necessariamente dello stesso tipo, ognuna identificata da un *nome*.

**struct
persona**



STRUTTURE

Definizione di una *variabile* di tipo *struttura*:

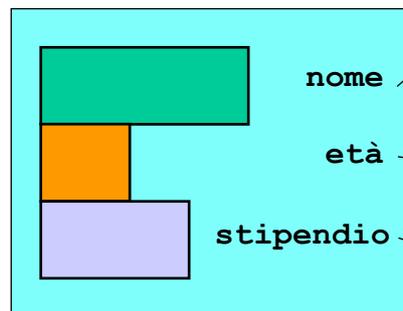
```
struct [<etichetta>] {  
    { <definizione-di-variabile> }  
} <nomeStruttura> ;
```

ESEMPIO

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

Definisce una variabile **pers** strutturata nel modo illustrato.

struct
persona



stringa
di 20
char

intero

float

ESEMPIO

```
struct punto {  
    int x, y;  
} p1, p2 ;
```

p1 e **p2** sono fatte
ciascuna da due interi
di nome **x** e **y**

```
struct data {  
    int giorno, mese, anno;  
} d ;
```

d è fatta da tre interi
di nome **giorno**,
mese e **anno**

STRUTTURE

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

Ad esempio:

```
p1.x = 10;  p1.y = 20;
```

```
p2.x = -1;  p2.y = 12;
```

```
d.giorno = 25;
```

```
d.mese = 12;
```

```
d.anno = 1999;
```

Ogni campo si usa come una normale variabile del tipo corrispondente al tipo del campo.

STRUTTURE

```
void main() {  
    struct persona{  
        char nome[20];  
        int  eta;  
        float stipendio;  
    } P1;  
    struct persona P2 ;  
    ...  
}
```



Non occorre ripetere l'elenco dei campi perché è *implicito* nell'etichetta *frutto*, che è già comparsa sopra.

ESEMPIO

```
void main() {
    struct persona {
        char nome[20];
        int  eta;
        float stipendio;
    } P1 ;
    struct persona P2;
    float sommastip;
    printf("Inserire nome, eta' e stipendio di P1);
    scanf("%s %d %f", P1.nome, &P1.eta, &P1.stipendio);

    printf("Inserire nome, eta' e stipendio di P2);
    scanf("%s %d %f", P2.nome, &P2.eta, &P2.stipendio);

    sommastip = P1.stipendio + P2.stipendio;
    printf("Lo stipendio totale e' %f", sommastip);
}
```

Non c'è alcuna ambiguità perché ogni variabile di nome `stipendio` è definita nella propria struct.

STRUTTURE

- A differenza di quanto accade con gli array, il nome della struttura rappresenta la struttura nel suo complesso.

Quindi, è possibile:

- assegnare una struttura a un'altra (`P2 = P1`)
- che una funzione restituisca una struttura

E soprattutto:

- passare una struttura come parametro a una funzione significa passare una copia

ASSEGNAZIONE TRA STRUTTURE

```
void main() {  
    struct persona {  
        char nome[20];  
        int  eta;  
        float stipendio;  
    } P1 ;  
    struct persona P2;  
    P2 = P1;  
}
```

Equivale a

```
P1.nome = P2.nome;  
P1.eta = P2.eta;  
P1.stipendio = P2.stipendio;
```

STRUTTURE PASSATE COME PARAMETRI

- Il nome della struttura rappresenta, come è naturale, *la struttura nel suo complesso*
- quindi, non ci sono problemi nel passarle a come parametro a una funzione: *avviene il classico passaggio per valore*
 - tutti i campi vengono copiati, uno per uno!
- è perciò possibile anche *restituire come risultato* una struttura

ESEMPIO

```
struct punto{float x,y;} P;
```

Tipo del valore di ritorno della funzione.



```
struct punto mezzo (  
    struct punto P1, struct punto P2) {  
    struct punto P;  
    P.x = (P1.x + P2.x) / 2;  
    P.y = (P1.y + P2.y) / 2;  
    return P;  
}
```

ESEMPIO

PROBLEMA: leggere le coordinate di un punto in un piano e modificarle a seconda dell'operazione richiesta:

proiezione sull'asse X
proiezione sull'asse Y
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

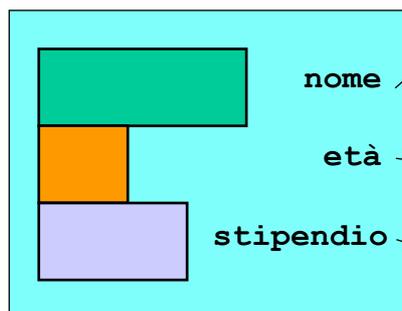
```
#include <stdio.h>
void main()
{ struct punto{float  x,y;} P;
  unsigned int op;
  float  Dx, Dy;
  printf("ascissa? ");   scanf("%f",&P.x);
  printf("ordinata? ");  scanf("%f",&P.y);
  printf("%s\n","operazione(0,1,2,3)?"); scanf("%d",&op);
  switch (op)
  {case 1: P.y= 0;break;
   case 2: P.x=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

DEFINIZIONE DI NUOVI TIPI

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

Definisce una variabile
pers strutturata nel
modo illustrato.

struct
persona



stringa
di 20
char
intero
float

TYPDEF

In C è possibile definire NUOVI TIPI.

```
typedef <descrizione_nuovo_tipo> NEWT;
```

- Si usa la parola chiave `typedef`
- La dichiarazione associa a un nuovo tipo di dato non primitivo un identificatore `NEWT`
- Le caratteristiche di nuovo tipo sono indicate in `<descrizione_nuovo_tipo>`

ESEMPIO

```
typedef int NewInt; /* NewInt è un tipo  
                    non primitivo che  
                    ridefinisce int */
```

```
NewInt X; /* X è di tipo NewInt */  
int Y;    /* Y è di tipo int */
```

Molto comodo per le strutture

ESEMPIO

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

Struct persona e' un tipo.

```
void main() {

    struct persona P1, P2;
    float sommastip;
    printf("Inserire nome, eta' e stipendio di P1);
    scanf("%s %d %f", P1.nome, &P1.eta, &P1.stipendio);

    printf("Inserire nome, eta' e stipendio di P2);
    scanf("%s %d %f", P2.nome, &P2.eta, &P2.stipendio);

    sommastip = P1.stipendio + P2.stipendio;

    printf("Lo stipendio totale e' %f", sommastip);
}
```

ESEMPIO

```
typedef struct {
    char nome[20];
    int eta;
    float stipendio;
} impiegato;
```

Da ora in poi, impiegato e' un nuovo TIPO definito dall'utente.

```
void main() {

    impiegato P1, P2;
    float sommastip;
    printf("Inserire nome, eta' e stipendio di P1);
    scanf("%s %d %f", P1.nome, &P1.eta, &P1.stipendio);

    printf("Inserire nome, eta' e stipendio di P2);
    scanf("%s %d %f", P2.nome, &P2.eta, &P2.stipendio);

    sommastip = P1.stipendio + P2.stipendio;

    printf("Lo stipendio totale e' %f", sommastip);
}
```