LE STRUCT

Tipo definito dall'utente i cui elementi possono essere ***eterogenei*** *(di* tipo diverso).

Introduce un **nuovo tipo di dato** definito dall'utente.

Definizione:

```
struct nome_struttura
    {
    Tipo   var1;
    Tipo   var2;
    ...
    Tipo   vark;
    };
```

oppure

```
typedef struct
        {
        Tipo   var1;
        Tipo   var2;
        ...
        Tipo   vark;
        } nome_struttura ;
```

Dichiarazione delle variabili di **tipo** nome_struttura:

nome_struttura var_struttura;

LE STRUCT

Dichiarazione di variabili di tipo struttura (altra possibilità) :

```
struct nome_struttura
    {
    tipo var1;
    tipo var2;
    ...
    tipo vark;
    } var_struttura1, var_struttura2;
```

oppure

```
struct
    {
    tipo var1;
    tipo var2;
    ...
    tipo vark;
    } var_struttura1, var_struttura2;
```

In questo caso nome_struttura non è necessario.

E' necessario solo se, successivamente, si vuole definire un'altra variabile dello stesso tipo:

nome_struttura var_struttura3;

LE STRUCT

Inizializzazione di una variabile di tipo struttura in fase di definizione o di dichiarazione.

Esempi:

```
struct   Cittadino
   { char Nome[20];
     char  Cognome[20];
     int Reddito;
     float Aliquota;
   } c1 ={"Valentina", "Florio", 7000+18000, 23.0 };
```

```
struct   Cittadino
   { char Nome[20];
     char  Cognome[20];
     int Reddito;
     float Aliquota;
   };

.....
Cittadino   c1 ={"Valentina", "Florio", 7000+18000, 23.0 };
Cittadino   c2 ={"", "Rossi",     , 18.0  };
```


LE STRUCT

Accesso ai campi: tramite l'operatore "**punto**".

var_struttura1.var1 = ...;

T = var_struttura1.var2; dove T è stata dichiarata dello stesso tipo del campo var2

Assegnamento di strutture: possibile l'assegnamento diretto tra record di tipo equivalente.

```
struct  s1
   {    int a, b;
   };

struct  s2
   {  int a, b;
   };
...

 s1  X, Y;
 s2  Z;
...
Y = X;      ! OK
Z = X;      ! ERRORE: non corrispondenza di tipo.
```

LE STRUCT

Confronto tra strutture: non possibile. Il seguente frammento di codice produce un errore:

```
if (X == Y)
...
```

Occorre considerare i singoli campi delle strutture:

```
if (X.a == Z.a && X.b == Z.b)
...
```

- **strutture diverse** possono avere **campi con lo stesso nome**;

- **i nomi dei campi** possono essere **gli stessi di variabili** e **funzioni** già utilizzate.

ARRAY di STRUCT (tabelle)

Si definisce una struttura e poi si dichiara un array di quel tipo.

Esempio:

```
struct   studente
   {  char nome[50];
      char cognome[50];
      int anno_immatricolazione;
   };

studente  elenco_studenti[100];
```

Per **accedere ad un record specifico** → **indicizzare il nome** della struttura:

```
cout << elenco_studenti[2].cognome;
```

stampa il cognome memorizzato nella variabile *cognome* dello studente i cui dati sono nella 3^ riga dell'array (l'indice dell'array parte sempre da 0).

ARRAY di STRUCT (tabelle)

Inizializzazione di una variabile di tipo tabella in fase di definizione.

Esempio:

```
struct   Contribuente
   { char Nome[20];
     char  Cognome[20];
     int Reddito;
     float Aliquota;
   };

…..
Contribuente  Contribuenti[20] ={ {"Valentina", "Florio", 7000+18000, 23.0 },
                                  {"", “Rossi",     , 18.0  },
                                  {“Alessio", “Verdi", 45000  , 27.0  },
                                  { },
                                  {“Marco", “Bianchi", 20500 , 20.5  };
                                 } ;
```


Esercizio

Crea un array di struct "studente“ (cognome, array contenente i voti degli esami). Presenta poi all’utente un menu’ con varie possibilita’:

1. media (in trentesimi);
2. media (in 110);
3. voto piu’ alto;
4. voto piu’ basso;
5. fine programma.

Realizza ciascuna operazione con una funzione.