

---

# Tipi aggregati



# Tipi aggregati

---

- In C, è possibile definire dati composti da elementi eterogenei (detti *record*), aggregandoli in una singola variabile
  - Individuata dalla keyword `struct`
- Sintassi (definizione di tipo):



```
struct <identificatore> {  
    campi  
};
```

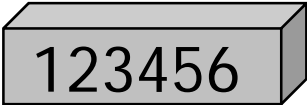
I campi sono nel formato:

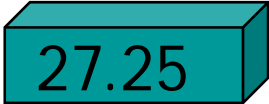
```
<tipo> <nome campo>;
```

## Perché dati eterogenei ?

- Sono frequenti le informazioni composte da parti di tipo diverso (es.: dati studente)

cognome  Rossi      nome  Mario

matri col a  123456

medi a  27.25

## Aggregato di dati eterogenei

- Più informazioni eterogenee possono essere unite come parti (campi) di uno stesso dato (dato aggregato)

studente

cognome: Rossi	
nome: Mario	
matricola: 123456	media: 27.25

## I tipi struct

---

- Il dato aggregato in C è detto `struct`. In altri linguaggi si parla di `record`
- Una `struct` (struttura) è un dato costituito da campi
  - I campi sono di tipi (base) noti (eventualmente altre `struct`)
  - Ogni campo all'interno di una `struct` è accessibile mediante un identificatore (anziché un indice, come nei vettori)

## Come creare un tipo struct

---

- Definire un tipo struct corrisponde a creare un nuovo tipo di dato, caratterizzato da
  - Parola chiave struct
  - Un nome (un identificatore) unico                      Es. struct studente
- I campi di una struct sono definiti alla stregua di variabili locali alla struct
- Le struct sono di solito dichiarate nell'intestazione di un file sorgente C
- Il nuovo tipo definito potrà essere utilizzato per dichiarare variabili o campi di altre strutture

# struct

---

- Una definizione di `struct` equivale ad una definizione di tipo
- Successivamente, una struttura può essere usata come un tipo per dichiarare variabili

- Esempio:

```
struct complex {  
    double re;  
    double im;  
}  
...  
struct complex num1, num2;
```

## Un esempio di tipo struct

---

```
struct studente
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
};
```



## Un esempio di tipo struct

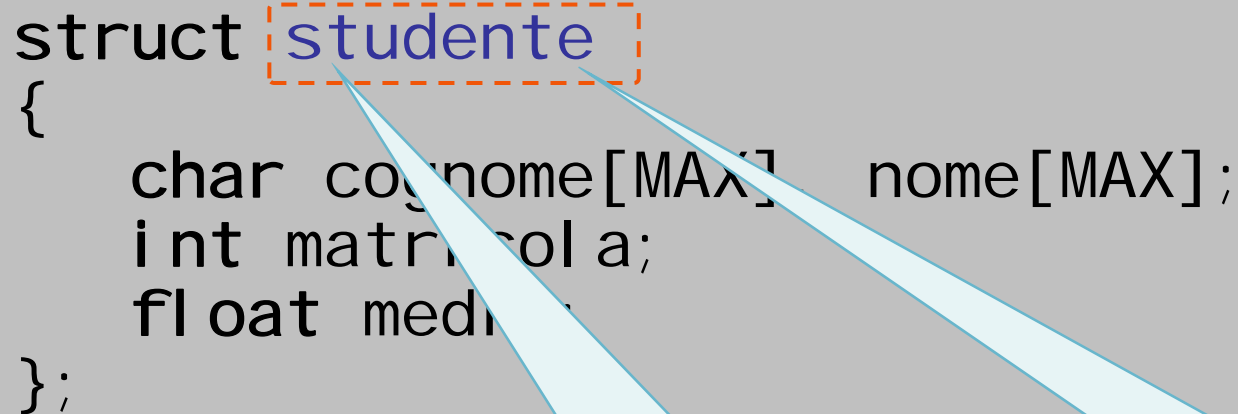
```
struct studente  
{  
    char cognome[MAX], nome[MAX];  
    int matricola;  
    float media;  
};
```

Nuovo tipo di  
dato

- Il nuovo tipo definito è `struct studente`
- La parola chiave `struct` è obbligatoria


## Un esempio di tipo struct

```
struct studente
{
    char cognome[MAX] nome[MAX];
    int matricola;
    float media;
};
```



Nuovo tipo di  
dato

Nome del tipo  
aggregato

- Stesse regole che valgono per i nomi delle variabili
  - I nomi di struct devono essere diversi da nomi di altre struct (possono essere uguali a nomi di variabili)
- 

## Un esempio di tipo struct

```
struct studente
```

```
{
```

```
    char cognome[MAX] nome[MAX];  
    int matricola;  
    float media;
```

```
};
```

Campi  
(eterogenei)

Nuovo tipo di  
dato

Nome del tipo  
aggregato

- I campi corrispondono a variabili locali di una struct
- Ogni campo è quindi caratterizzato da un tipo (base) e da un identificatore (unico per la struttura)

## Dichiarazione di variabili struct

```
struct studente  
{  
    char cognome[MAX], nome[MAX];  
    int matricola;  
    float media;  
};
```

```
struct studente s, t;
```

Dichiarazione delle variabili **s** e **t** di  
tipo **struct studente**

## struct: Esempi

---

```
struct complex {  
    double re;  
    double im;  
}
```

```
struct identity {  
    char nome[30];  
    char cognome[30];  
    char codicefiscale[15];  
    int altezza;  
    char statocivile;  
}
```

## Accesso ai campi di una struct

---

- Una struttura permette di accedere ai singoli campi tramite l'operatore '.', applicato a variabili del corrispondente tipo struct

*<variabile> . <campo>*

- Esempio:

```
struct complex {  
    double re;  
    double im;  
}  
...  
struct complex num1, num2;  
num1.re = 0.33; num1.im = -0.43943;  
num2.re = -0.133; num2.im = -0.49;
```

## Schemi di dichiarazione alternativi

### 1. Schema proposto

```
struct studente
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
};
...
struct studente s, t;
```

## Schemi di dichiarazione alternativi

### 2. Dichiarazione/definizione contestuale di tipo struct e variabili

- Tipo struct e variabili vanno definiti nello stesso contesto (globale o locale)

```
struct studente
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
} s, t;
```



# Definizione di struct come tipo

---

- E' possibile definire un nuovo tipo a partire da una struct tramite la direttiva `typedef`
  - Passabile come parametro
  - Indicizzabile in vettori

- Sintassi:

`typedef <tipo> <nome nuovo tipo>;`

- Esempio:

```
typedef struct complex {  
    double re;  
    double im;  
} compl;  
                        → compl z1,z2;
```

## Definizione di struct come tipo (Cont.)

---

- Passaggio di struct come argomenti

```
int f1 (compl z1, compl z2)
```

- struct come risultato di funzioni

```
compl f2(.....)
```

- Vettore di struct

```
compl lista[10];
```

- Nota:

La direttiva `typedef` è applicabile anche non alle strutture per definire nuovi tipi

- Esempio: `typedef unsigned char BIT8;`

## Schemi di dichiarazione alternativi

4. Sinonimo di `struct studente` introdotto mediante `typedef`

```
typedef struct studente
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
} Studente;
...
Studente s, t;
```

## Schemi di dichiarazione alternativi

5. Sinonimo introdotto mediante typedef: variante senza identificatore di struct

```
typedef struct studente
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
} Studente;
...
Studente s, t;
```

Identificatore inutilizzato

## Schemi di dichiarazione alternativi

5. Sinonimo introdotto mediante typedef: variante senza identificatore di struct

```
typedef struct
{
    char cognome[MAX], nome[MAX];
    int matricola;
    float media;
} Studente;

...
Studente s, t;
```

# Operazioni su struct

---

- Confronto:

- Non è possibile confrontare due variabili dello stesso tipo di `struct` usando il loro nome

- Esempio:

- `comp1 s1, s2 ➡ s1==s2`    o   `s1!=s2` è un errore di sintassi

- Il confronto deve avvenire confrontando i campi uno ad uno

- Esempio:

- `comp1 s1, s2 ➡ (s1.re == s2.re) && (s1.im == s2.im)`

- Inizializzazione:

- Come per i vettori, tramite una lista di valori tra { }

- Esempio:

- `comp1 s1 = {0.1213, 2.655};`

- L'associazione è posizionale: In caso di valori mancanti, questi vengono inizializzati a:

- 0, per valori "numerici"
  - NULL per puntatori

# Esercizio 1

---

- Data la seguente struct :

```
struct stud {  
    char nome[40];  
    unsigned int matricola;  
    unsigned int voto;  
}
```

- Si definisca un corrispondente tipo `studente`
- Si scriva un `main()` che allochi un vettore di 10 elementi e che invochi la funzione descritta di seguito
- Si scriva una funzione `ContaInsufficienti()` che riceva come argomento il vettore sopracitato e ritorni il numero di studenti che sono insufficienti

# Esercizio 1: Soluzione

---

```
#include <stdio.h>
```

```
#define NSTUD 10
```

```
typedef struct stud {  
    char nome[40];  
    unsigned int matricola;  
    unsigned int voto;  
} studente;
```

```
int ContaInsufficienti(studente vett[], int dim); /*  
    prototipo */
```



## Esercizio 1: Soluzione (Cont.)

---

```
main()
{
    int i, NumIns;
    studente Lista[NSTUD];

    /* assumiamo che il programma riempia
       con valori opportuni la lista */

    NumIns = ContaInsufficienti(Lista, NSTUD);
    printf("Il numero di insufficienti e': %d.\n", NumIns);
}

int ContaInsufficienti(studente s[], int numstud)
{
    int i, n=0;

    for (i=0; i<numstud; i++) {
        if (s[i].voto < 18)
            n++;
    }
    return n;
}
```

## Esercizio 2

---

- Data una struct che rappresenta un punto nel piano cartesiano a due dimensioni:

```
struct point {  
    double x;  
    double y;  
};
```

ed il relativo tipo `typedef struct point Point;` scrivere le seguenti funzioni che operano su oggetti di tipo `Point`:

- `double DistanzaDaOrigine (Point p);`
- `double Distanza (Point p1, Point p2);`
- `int Quadrante (Point p);` /\* in quale quadrante \*/
- `int Allineati(Point p1, Point p2, Point p3);`  
/\* se sono allineati \*/
- `int Interseca(Point p1, Point p2);`  
/\* se il segmento che ha per estremi p1 e p2  
interseca un qualunque asse\*/

## Esercizio 2: Soluzione

---

```
double DistanzaDaOrigine (Point p)
{
    return sqrt(p.x*p.x + p.y*p.y);
}
```

```
double Distanza (Point p1, Point p2)
{
    return sqrt((p1.x-p2.x)*(p1.x-p2.x) +
                (p1.y-p2.y)*(p1.y-p2.y));
}
```

```
int Quadrante (Point p)
{
    if (p.x >= 0 && p.y >= 0) return 1;
    if (p.x <= 0 && p.y >= 0) return 2;
    if (p.x <= 0 && p.y <= 0) return 3;
    if (p.x >= 0 && p.y <= 0) return 4;
}
```

## Esercizio 2: Soluzione (Cont.)

---

```
int Allineati (Point p1, Point p2, Point p3)
{
    double r1, r2;
    /* verifichiamo che il rapporto tra y e x
       delle due coppie di punti sia identico */
    r1 = (p2.y-p1.y)/(p2.x-p1.x);
    r2 = (p3.y-p2.y)/(p3.x-p2.x);
    if (r1 == r2)
        return 1;
    else
        return 0;
}

int Interseca(Point p1, Point p2)
{
    /* verifichiamo se sono in quadranti diversi */
    if (Quadrante(p1) == Quadrante(p2))
        return 0;
    else
        return 1;
}
```