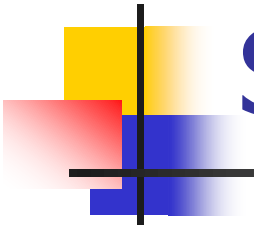
Tipo Struct, Union, Enum

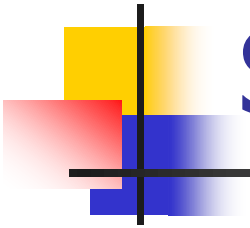
Concetti chiave

- Variabili strutturate: il tipo struct
- Elementi di una struttura
- typedef & struct
- Strutture annidate
- Array di strutture
- Union
- Enum



Struct

- Per *aggregare* elementi di tipo diverso si utilizza un tipo di dato derivato chiamato **struct**.
- E' una sequenza di elementi detti membri o campi (fields), che occupano in memoria posizioni sequenziali e che possono essere di tipo differente



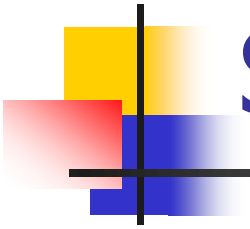
Struct

- La sintassi per la dichiarazione di una nuova struttura è:

```
struct nomeStruttura  {
    tipoMembro nomeMembro1;
    tipoMembro nomeMembro2;
    ...
    tipoMembro nomeMembroN;
};
```

- Una volta definito il tipo struct, si possono definire variabili di quel tipo:

```
struct nomeStruttura nomeVariabile;
```



Struct

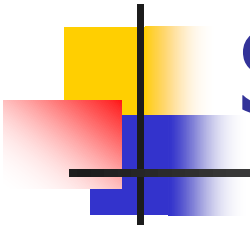
- E' possibile anche la seguente dichiarazione:

```
struct {
    tipoMembro nomeMembro1;
    tipoMembro nomeMembro2;
    ...
    tipoMembro nomeMembroN;
} nomeVariabile;
```

Struct

- Esempi:

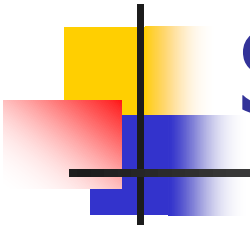
```
struct automobile {
   char marca[10];
   char modello[10];
   unsigned int  prezzo;
}  a1, a2;

struct automobile {
   char marca[10];
   char modello[10];
   unsigned int  prezzo;
};
struct automobile a1, a2;

struct {
   char marca[10];
   char modello[10];
   unsigned int  prezzo;
} a1, a2;
```



Struct

- Inizializzazione:

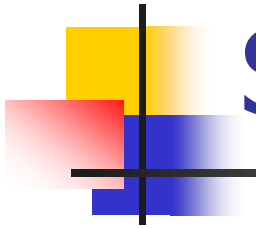
```
struct [opzionale nomeStruct]{
    tipoMembro nomeMembro1;
    tipoMembro nomeMembro2;
    ...
    tipoMembro nomeMembroN;
} nomeVariabile ={elenco valori};
```



Struct

- Inizializzazione:

```
struct [opzionale nomeStruct]{
                tipoMembro nomeMembro1;
                tipoMembro nomeMembro2;
                ...
                tipoMembro nomeMembroN;
 }nomeVariabile={.nomeMembro=value, .nomeMembro=value};
```



Struct

- Esempio di Inizializzazione:

```
struct {
        char cognome[20];
        unsigned int eta;
} x = { "rossi",20 }, y = { .eta = 10 };
```

Struct

- Operatore = applicato alle struct

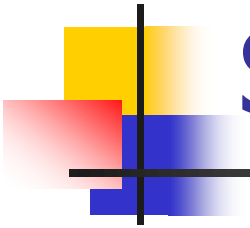
```
struct automobile {
   char marca[10];
   char modello[10];
   int  prezzo;
}  a1, a2;
```

```
a1=a2;
```

- Viene effettuata la copia membro per membro
- Le variabili devono appartenere allo stesso tipo

```
struct auto1 {
   char marca[10];
   char modello[10];
   int  prezzo;
}  a1;
```

```
struct auto2 {
      char marca[10];
      char modello[10];
      int  prezzo;
} a2;
```



Struct

- Per riferirsi a un membro ***nomeVariabileStruttura.nomeMembro***

```
struct TriangoloRettangolo {
  float cateto1, cateto2, area;
} a;

scanf("%f",&a.cateto1);
scanf("%f",&a.cateto2);
a.area=a.cateto1*a.cateto2/2;
printf("\nArea del Triangolo = %f ",a.area);
```

Struct

- Per riferirsi a un membro ***nomeVariabileStruttura.nomeMembro***

```c
#include <stdio.h>
#define S 30
#define CF 17

struct {
          char cognome[S], nome [S], codice_fiscale[CF];
          unsigned int eta;
} x;

int main(void)
{
   strcpy(x.cognome, "rossi");
   strcpy(x.nome, "mario");
   strcpy(x.codice_fiscale, "RSSMRA65D30H163O");
   x.eta = 51;

   printf("\nCognome = %s ",x.cognome);
   printf("\nNome = %s ", x.nome);
   printf("\nCodice Fiscale = %s ", x.codice_fiscale);
   printf("\nEta' = %u ", x.eta);
}
```

Struct

- Quanto spazio occupano le variabili di tipo struct ?
- Risposta che sembra ovvia: somma delle dimensioni dei membri.

```
#include <stdio.h>

struct S1 {
unsigned long u;
float d;
};

int main(void)
{
      printf("\nDimensione del tipo Struct S1 = %zu ", sizeof(struct S1));
      printf("\n");
}
```

Dimensione = 8 byte

Struct

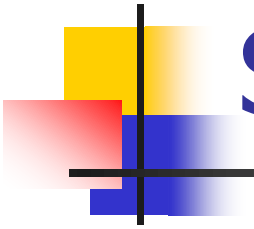
- Quanto spazio occupano le variabili di tipo struct ?
- Risposta che sembra ovvia: somma delle dimensioni dei membri.
- **Purtroppo non è sempre così**

```
#include <stdio.h>

struct S2 {
    char c_1;
    double d;
    char c_2;
};

int main(void)
{
     printf("\n%zu ", sizeof(struct S2));
}
```

Dimensione = 24 byte

```
#include <stdio.h>

struct S3 {
   double d;
   char c_1;
   char c_2;
};

int main(void)
{
    printf("\n%zu ", sizeof(struct S3));
}
```

Dimensione = 16 byte



Struct

- Quanto spazio occupano le variabili di tipo struct ?
- Risposta che sembra ovvia: somma delle dimensioni dei membri.
- **Purtroppo non è sempre così**
- **A volte, dentro uno struct, vengono lasciati degli spazi vuoti per rendere più facile l'accesso in memoria (ad esempio per fare in modo che tutti i membri partano da indirizzi pari)**
- **Regola generale: dichiarare in membri in ordine decrescente di dimensione (prima i tipi che occupano più spazio)**

Strutture Annidate
Struct di Array

```
struct persona {
   char nome[30];
   char cognome[30];
   char comuneNasc[30];
   char telefono[10];
   char parentela[2];
};
```

La struttura persona ha dei membri che sono array.

Strutture Annidate
Struct di Struct

```
struct data {
    int giorno;
    char mese [20];
    int anno;
};

struct persona {
    char nome[30];
    char cognome[30];
    struct data dataNasc;
    char comuneNasc[30];
    char telefono[10];
    char parentela[2];
};
```

La struttura `persona` ha dei membri che sono a loro volta delle variabili struttura.

Strutture Annidate
Struct di Struct

```
#include<stdio.h>

struct data {
    unsigned short giorno;
    char mese[20];
    unsigned short anno;
};

struct persona {
      char nome[30];
      char cognome[30];
      struct data dataNasc;
      char comuneNasc[30];
      char telefono[10];
      char parentela[2];
} p1, p2;
```

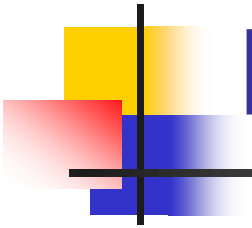
```
int main(void)
{
   printf("\nInserisci il Cognome ");
   fgets(p1.cognome, 30, stdin);
   printf("\nCognome = %s ", p1.cognome);
   printf("\nInserisci il giorno di nascita ");
   scanf("%hu", &p1.dataNasc.giorno);
   printf("\nGiorno di Nascita %u", p1.dataNasc.giorno);
   strcpy(p1.dataNasc.mese, "Gennaio");
   printf("\nMese di Nascita %d", p1.dataNasc.giorno);
   p1.dataNasc.anno = 2013;
   printf("\nAnno di Nascita %d", p1.dataNasc.giorno);
}
```

Typedef

```
typedef struct {
   unsigned short giorno;
   char mese [20];
   unsigned short anno;
} data ;


typedef struct {
   char nome[30];
   char cognome[30];
   data dataNasc;
   char comuneNasc[30];
   char telefono[10];
} persona;

data x,y,z;
persona a,b,c;
```

Strutture Annidate
Array di Struct

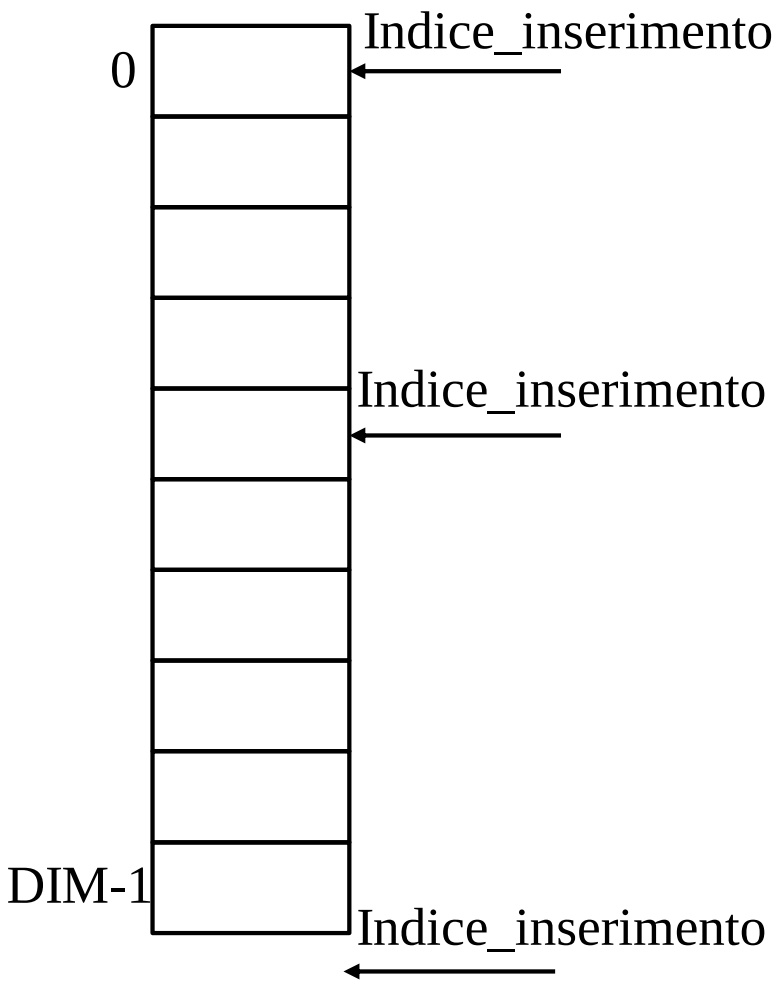
```
#include<stdio.h>
#define N 10

typedef struct {
      char cognome[30], nome[30], telefono[15];
} persona;

persona vettore[N];

int main(void) {
    printf("\nInserisci il Cognome ");
    fgets(vettore[0].cognome, 30, stdin);
    printf("\nCognome = %s ", vettore[0].cognome);
}
```



Esercizio

- Programma per la gestione di un vettore di struct (cognome, nome, telefono)
- Riempimento di un elemento alla volta
- Visualizzazione di tutto l'archivio
- Ricerca per cognome e nome e visualizzazione del telefono

Svolgimento 1/4

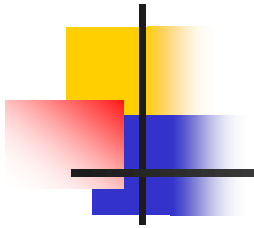



```
#include<stdio.h>
#include<string.h>

#define FFLUSH while(getchar()!='\n')
#define DIM 10
#define N 30
#define T 15

typedef struct {
    char cognome[N], nome[N], telefono[T];
} persona;

persona vettore[DIM], tmp;
unsigned long indice_inserimento, i, j;
unsigned short scelta,l;

int main(void)
{
   indice_inserimento = 0;
   do {
       printf("\n1)Inserimento ");
       printf("\n2)Visualizzazione ");
       printf("\n3)Ricerca per cognome e nome ");
       printf("\n4)FINE ");
       scanf("%hu", &scelta);
       FFLUSH;
```

Svolgimento 2/4

```
case 1:   if (indice_inserimento == DIM) printf("\nArchivio Pieno ");
          else {
              printf("\nInserisci il Cognome ");
              fgets(vettore[indice_inserimento].cognome,N,stdin);
              l = strlen(vettore[indice_inserimento].cognome);
              if (l < N - 1) vettore[indice_inserimento].cognome[l - 1] = '\0';
              printf("\nInserisci il Nome ");
              fgets(vettore[indice_inserimento].nome,N,stdin);
              l = strlen(vettore[indice_inserimento].nome);
              if (l < N - 1) vettore[indice_inserimento].nome[l - 1] = '\0';
              printf("\nInserisci il Telefono ");
              fgets(vettore[indice_inserimento].telefono,T,stdin);
              l = strlen(vettore[indice_inserimento].telefono);
              if (l < T - 1) vettore[indice_inserimento].telefono[l - 1] = '\0';

              indice_inserimento++;
          }
          break;
```

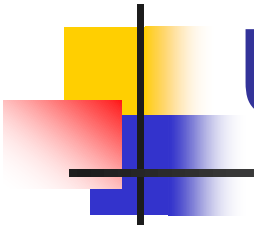
Svolgimento 3/4



```
case 2: if (indice_inserimento==0) printf("\nArchivio Vuoto ");
        else for (i=0; i<indice_inserimento; i++) {
                printf("\nCognome = %s ",vettore[i].cognome);
                printf("\nNome = %s ",vettore[i].nome);
                printf("\nTelefono = %s ",vettore[i].telefono);
        }
        break;
```

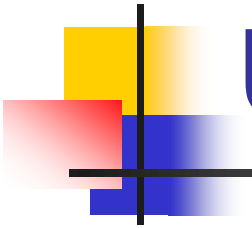
Svolgimento 4/4

```
    case 3: if (indice_inserimento == 0) printf("\nArchivio Vuoto ");
                else {
                    printf("\nInserisci il Cognome ");
                    fgets(tmp.cognome,N,stdin);
                    l = strlen(tmp.cognome);
                    if (l < N - 1) tmp.cognome[l - 1] = '\0';
                    printf("\nInserisci il Nome ");
                    fgets(tmp.nome,N,stdin);
                    l = strlen(tmp.nome);
                    if (l < N - 1) tmp.nome[l - 1] = '\0';
                    for (i = 0; i<indice_inserimento; i++)
                        if (!strcmp(vettore[i].cognome, tmp.cognome) && !strcmp(vettore[i].nome, tmp.nome))
                       {
                              printf("\nElemento Trovato ");
                              printf("\nIl Telefono e' %s ", vettore[i].telefono);
                              break;
                        }
                    if (i == indice_inserimento) printf("\nNON ESISTE ");
                }
  }
 } while (scelta<4);
}
```



Union

- Union è un tipo di dato derivato
- E' una sequenza di elementi di tipo diverso, detti membri o fields
- Non vengono allocati in modo sequenziale come gli struct, ma in modo "sovrapposto"
- Può essere memorizzato uno solo tra i membri
- La dimensione occupata coincide con la dimensione più grande tra i fields
- Vantaggio principale: riduzione di uso di memoria in alcuni scenari



Union

- Sintassi della dichiarazione:

```
union nomeUnion [opzionale] {
    tipo_dato nomeMembro1;
    tipo_dato nomeMembro2;

    tipo_dato nomeMembro3;
} nomeVariabile [opzionale];
```

Union

- Nella dichiarazione è possibile utilizzare typedef come per gli struct
- Nella dichiarazione è possibile inizializzare, ma solo un membro, per default è sempre il primo
- Esempio:

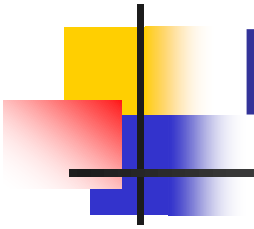
```
union U {
    char c;
    int i;
} u={'a'};
u.i=100; /*u.c non esisterà più*/
```

```
union U {
    char c;
    int i;
} u={.i=500};
```

Union

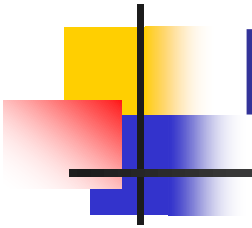
- Vantaggio delle Union: riduzione memoria in alcuni casi
- Nell'esempio, si vuole creare un archivio dove per ogni persona si memorizza in ogni caso cognome e nome
- Per ogni persona di possono memorizzare in modo alternativo o i dati relativi all'IVA o al Codice Fiscale
- Se non ci fosse Union, bisognava creare un vettore di elementi che contenevano tutte le informazioni, che potevano essere non usate

```
typedef struct {
  char codice_fiscale[CF];
  char residenza[N];
} CoF;


typedef struct {
  char partita_IVA[PI];
  char ragione_sociale[N];
} IVA;

typedef union {
  CoF cf;
  IVA iva;
} fisco;

typedef struct {
  char cognome[N], nome[N];
  fisco dati;
} cliente;


cliente vettore[DIM];
```



Enum

- Un'enumerazione è un tipo di dato intero rappresentato da una serie di nomi o identificativi simbolici costanti di tipo int
- Si dichiara come le struct o le union
- Si usano per definire dei nomi significativi a ciascuno dei quali si associa un intero
- Di fatto è come se si definissero dei simboli #define



Enum

- Sintassi della dichiarazione:

```
enum nomeEnum [opzionale] {
    identificatore1 [=valore],
    identificatore2 [=valore],

    identificatoreN [=valore]
} nomeVariabile [opzionale];
```

Enum

- Esempio:
- Dall'esempio si capisce che ogni identificatore ha associato per default un numero intero a partire da zero
- Se si associa un valore, il successivo identificatore ha associato un numero intero +1

```
#include<stdio.h>

typedef enum {
    Red,
    Green=100,
    Blue
} RGB;

RGB x;

int main(void)
{
   x = Red;
   printf("\nValore x = %d ", x);
   x = Green;
   printf("\nValore x = %d ", x);
   printf("\nValore x = %d ", ++x);

}
```

Enum

Esempio:

- Uso degli Enum per realizzare menu di scelta

```c
#include<stdio.h>

typedef enum {
    Inserisci=1,
    Ricerca,
    Fine
} choice;

choice scelta;

int main(void)
{
  do {
    printf("\n1)Inserisci ");
    printf("\n2)Ricerca ");
    printf("\n3)FINE ");
    scanf("%d", &scelta);
    switch (scelta) {
       case Inserisci: printf("\nUNO"); break;
       case Ricerca: printf("\nDUE"); break;
       case Fine: printf("\nFINE");
    }
  } while (scelta != Fine);
}
```