

Strutture

Una ***struttura*** è una ***collezione finita di dati anche eterogenei*** (non necessariamente dello stesso tipo), ognuna identificata da un ***nome***

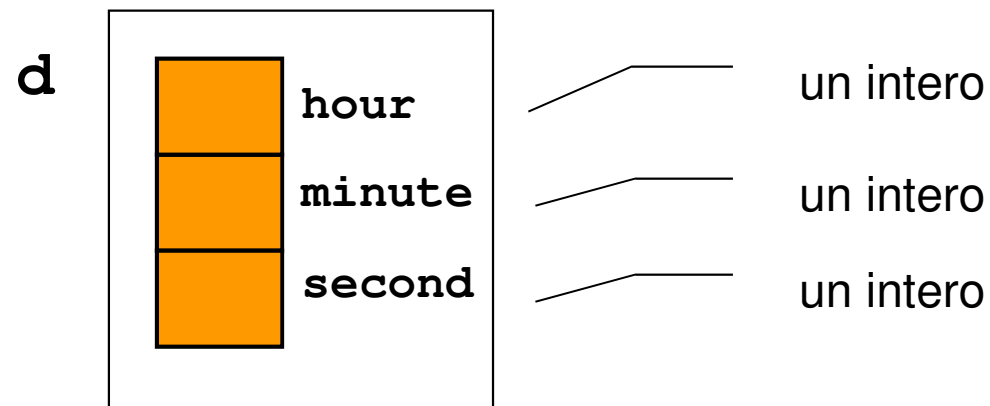
Definizione di una *variabile* di tipo ***struttura***:

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

Strutture

```
struct time
{
    int hour, minute, second;
} t ;
```

La variabile **t** è composta da tre interi di nome **hour**, **minute** e **second**



Strutture

- `time` è solo un'etichetta, è opzionale e serve per dichiarare altre variabili dello stesso tipo
- `struct time t1, t2;`
→ dichiara due variabili `t1` e `t2` di tipo struttura `time`
- L'accesso ai campi delle strutture avviene tramite la notazione puntata:

```
t1.hour = 12;  
t1.minute = 55;  
t1.second = 23;  
printf("It's %d:%d:%d; it's time for LUNCH!",  
      t1.hour, t1.minute, t1.second);
```

Strutture

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

È possibile:

■ ***assegnare una struttura a un'altra (copia!)***

- `f2 = f1;`

■ ***che una funzione restituisca una struttura (restituzione di una copia!)***

- ```
struct time getNoon()
{
 struct time t;
 t.hour = 12; t.minute = 0; t.second = 0;
 return t;
}
```

■ ***passare una struttura come parametro a una funzione (passaggio di una copia!)***

# Strutture & Array: piccolo trucco

---

Se una struttura, anche molto voluminosa, viene copiata elemento per elemento...

.. *perché non usare una struttura per incapsulare un array?*

In effetti:

- il C non rifiuta di manipolare gli array come un tutt'uno “per principio”: è solo la conseguenza del modo in cui si interpreta il loro nome
- quindi, *“chiudendoli in una struttura”...*

# Strutture & Array

---

```
int main() {
 struct string20
 {
 char s[20];
 }

 s1 = {"Paolino Paperino" },
 s2 = {"Gastone Paperone" };

 s1 = s2; /* FUNZIONA! */
}
```

- È fondamentale ricordare che si stanno assegnando strutture che contengono array e non array direttamente

# Strutture & Array

---

Analogamente, adottando lo stesso “trucco”, una funzione può essere forzata a restituire un array come valore di ritorno:

```
struct string20 { char s[20]; } ;
struct string20 maiusc(struct string20 x)
{
 int k;
 for (k = 0; k < strlen(x.s); k++)
 x.s[k] = toupper(x.s[k]);
 return x;
}
int main()
{
 struct string20 m = {"Che bello!"}, mm;
 mm = maiusc(m);
 printf("%s", mm.s);
}
```

# Esercizio

---

- Sia data la struttura

```
struct time
{
 int hour, minute, second;
};
```

- Si progetti una funzione in grado di calcolare la differenza fra due strutture `time` e che restituisca il risultato in termini di una nuova struttura `time`



# Esercizio

---

- Per semplicità si può definire il tipo Time

```
typedef struct time Time;
```

- L'interfaccia della funzione è facilmente desumibile dalle specifiche:

```
Time subtract(Time t1, Time t2);
```

- Due possibili approcci:

1. Trasformare in secondi, eseguire la differenza, trasformare in ore, minuti, secondi
2. Eseguire la sottrazione direttamente tenendo conto dei riporti

# Esercizio

---

```
Time subtract1(Time t1, Time t2)
{
 int s1, s2, sResult;
 Time result;

 s1 = t1.hour * 3600 + t1.minute * 60 + t1.second;
 s2 = t2.hour * 3600 + t2.minute * 60 + t2.second;
 sResult = s1 - s2;

 result.hour = sResult / 3600;
 sResult = sResult % 3600;
 result.minute = sResult / 60;
 sResult = sResult % 60;
 result.second = sResult;

 return result;
}
```

# Esercizio

---

```
Time subtract2(Time t1, Time t2)
{
 Time result;
 int carry;
 result.second = t1.second - t2.second;
 carry = 0;
 if (result.second < 0)
 {
 result.second = 60 + result.second;
 carry = -1;
 }
 result.minute = t1.minute - t2.minute + carry;
 carry = 0;
 if (result.minute < 0)
 {
 result.minute = 60 + result.minute;
 carry = -1;
 }
 result.hour = t1.hour - t2.hour + carry;
 return result;
}
```

# Strutture innestate

---

- Ovviamente (?) non ci sono problemi ad ***innestare strutture in altre strutture***
- Ad esempio si può pensare di avere una struttura *address* contenuta nella struttura *person*
- Come esercizio si può pensare di fornire alcune funzioni (servizi) che consentano di operare in modo agevole con le strutture di cui sopra
- Per cominciare:
  - Operazioni di lettura da console
  - Operazioni di formattazione su stringa

# Person & Address – Definizioni

---

```
typedef struct addressStruct
{
 char street[80];
 char postalCode[8];
 char city[30];
 char state[20];
} Address;
```

```
typedef struct personStruct
{
 char firstName[50];
 char secondName[50];
 char phone[18];
 char cell[18];
 Address address;
} Person;

#define PERSONARRAYDIM 100

typedef Person
 PersonArray[PERSONARRAYDI]
 ;
```

# Person & Address – Ricerca Esatta

---

## *Ricerca di un contatto per cognome (first name)*

- Problema facile e già visto
- Se i contatti sono:
  - ordinati → ricerca binaria
  - non ordinati → ricerca lineare
- Per semplicità si implementa la ricerca lineare...
- Si può utilizzare `strcmp()` come funzione di confronto fra stringhe...

# Person & Address – Ricerca Esatta

---

In ingresso:

- cognome da cercare
- array in cui cercare
- numero di strutture effettivamente presenti nell'array

```
int findExactByFirstName(char firstName[50],
 PersonArray persons, int dim)
{
 int i;
 for (i = 0; i < dim; i++)
 if (strcmp(persons[i].firstName, firstName) ==
0)
 return i;
 return -1;
}
```