

Typedef e Enum

Prof. Salvatore Venticinque

Prof. Mario Magliulo

typedef

- **typedef** può essere utilizzato per rinominare *qualsiasi* tipo
- Esempio,
 - `typedef char * String;`
- Esempi,
 - `typedef int size_t;`
 - `typedef long int32;`
 - `typedef long long int64;`

Tipo enum

- Un tipo enumerato viene specificato tramite l'elenco dei valori che i dati di quel tipo possono assumere:

```
typedef enum {a1, a2, a3, ... , an} EnumType;
```

- Il compilatore associa a ciascun identificatore del dominio un numero naturale, che viene utilizzato nella valutazione di espressioni, relazioni ed assegnazioni.
- stessa occupazione, stesso range e stessi operatori di **int**
/* a1, a2, a3,..., an usabili come costanti */

```
EnumType var1,var2;
```

```
var1 = a3;
```

Typo enum

- **Esempi:**

```
typedef enum {lu, ma, me, gi, ve, sa, do} Giorni;  
typedef enum {cuori, picche, quadri, fiori} Carte;  
Carte C1, C2, C3, C4, C5;  
Giorni Giorno;  
if (Giorno == do)  
    /* giorno festivo */  
else  
    /* giorno feriale */
```

- *L'utilizzo di tipi ottenuti per enumerazione rende più leggibile il codice.*

Typo enum

- Un identificatore di un valore scalare definito dall'utente deve comparire nella definizione di un solo tipo enumerato.

```
typedef enum {lu, ma, me, gi, ve, sa, do} Giorni;
```

```
typedef enum {lu, ma, me} PrimiGiorni;
```

- *Un tipo enumerato è totalmente ordinato* (Integral Type).

lu < ma → vero

lu >= sa → falso

cuori < quadri → vero

- E' anche possibile specificare un valore naturale a cui associare i singoli simboli:

```
typedef enum {gen, feb, mar, ...} Mesi;
```

gen → 0, feb → 1, etc

```
typedef enum {gen=1, feb, mar, ...} Mesi;
```

gen → 1, feb → 2, etc

```
typedef enum {gen=1, feb=4, mar, ...} Mesi;
```

gen → 1, feb → 4, mar → 5, etc

ATTENZIONE:

Non c'è controllo sugli estremi!

m1 = 15;

viene accettato ed eseguito

Struct e Union

Prof. Salvatore Venticinque

Prof. Mario Magliulo

Introduzione

- Gli array sono uno tipo di dato strutturato per il trattamento di insiemi di variabili di uno stesso tipo.
- Quando i tipi di dati, che devono essere logicamente aggregati, sono distinti, è necessario usare un diverso tipo strutturato: la **struct** (o record)

Introduzione

- **Esempio:** *Supponiamo di voler memorizzare, relativamente a ciascun abitante di un dato comune, nome e cognome, data di nascita e codice fiscale:*
- Nome e cognome costituiscono un array di caratteri
- La data è composta da tre numeri interi, che descrivono giorno, mese ed anno
- Il codice fiscale è un array di 16 caratteri (15 per il codice ed uno per il carattere nullo di terminazione)
- Le informazioni non possono essere collezionate in un unico array di caratteri, perché sono disomogenee

Definizione di una struct

- Definisce un nuovo tipo:
 - Per il compilatore è un'unità di informazione
- Esempio:

```
struct motor {  
    float volts;           //voltage of the motor  
    float amps;           //amperage of the motor  
    int phases;           //# of phases of the motor  
    float rpm;            //rotational speed of motor  
};                        //struct motor
```

La struct in C

- La **struct** è un costrutto del linguaggio C che consente la definizione di una struttura dati
- costituita da **campi** (piuttosto che da elementi)
- che sono identificati da **nomi** (piuttosto che da indici)
- e possono contenere informazione di tipo diverso

struct

- Definizione del nuovo tipo

- Esempio,

```
struct motor {  
    float volts;  
    float amps;  
    int phases;  
    float rpm;  
};          //struct motor
```

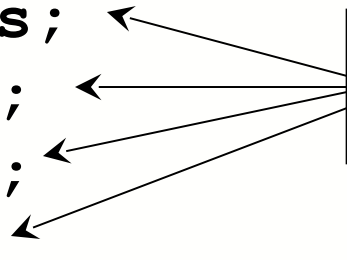
Nome del tipo

Nota:– il nome è opzionale solo se si definisce una sola **struct**

struct

- Definizione del nuovo tipo
- Esempio,

```
struct motor {  
    float volts;  
    float amps;  
    int phases;  
    float rpm;  
};          //struct motor
```



Membri o campi
della **struct**

Dichiarazione di variabili `struct`

```
struct motor p, q, r;
```

- Dichiarare e allocare spazio per tre variabili – `p`, `q`, e `r` – ognuna di tipo `struct motor`

```
struct motor M[25];
```

- Dichiarare un array di 25 elementi di tipo `struct motor`; alloca 25 unità di spazio, ognuna tale da contenere i dati di una struttura `motor`

```
struct motor *m;
```

- Dichiarare un puntatore a una variabile `struct motor`, ma non lo spazio necessario a contenerla

Utilizzo dei membri di una `struct`

- Se dichiariamo

```
struct motor p;  
struct motor q[10];
```

- Allora

<code>p.volts</code>	— è il voltaggio (variabile float)
<code>p.amps</code>	— è l'amperaggio (variabile float)
<code>p.phases</code>	— il numero di fasi (int)
<code>p.rpm</code>	— la velocità di rotazione (float)

`q[i].volts` — il voltaggio dell'**i-esimo** motor

`q[i].rpm` — velocità dell'**i-esimo** motor

Utilizzo dei campi di una **struct**

- Sia

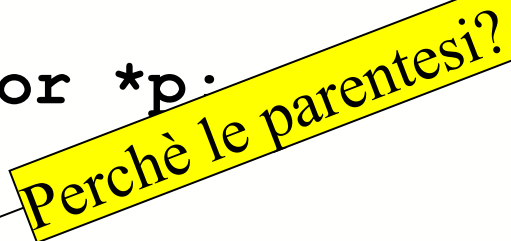
`struct motor *p;`

- Allora

`(*p).volts` — il valore del campo volts della struttura `motor` puntata da `p`

`(*p).phases` — il numero di fasi della struttura `motor` puntata `p`

Perchè le parentesi?



Utilizzo dei campi di una **struct**

- Sia

`struct motor`

- Allora

`(*p).volts` — il valore del campo `volts` della struttura `motor` puntata da `p`

`(*p).phases` — il numero di fasi della struttura `motor` puntata `p`

L'operatore `'.'` ha precedenza rispetto a `'*'`

Utilizzo dei campi di una **struct**

- Sia

```
struct motor *p;
```

- Allora

(*p) .volts — il valore del campo volts della struttura **motor** puntata da **p**

(*p) .phases — il numero di fasi della struttura **motor** puntata **p**

Utilizzo dei campi di una **struct**

- La notazione **(*p) .member** è scomoda
 - Fastidiosa da digitare; occorre aprire e chiudere ()
 - Molti tasti da digitare
- Il frequente utilizzo ha portato a introdurre la seguente notazione:
 - **p->member**, dove **p** è un puntatore a variabile struct

Il precedente esempio diventa:

- Sia:
`struct motor *p;`
- Allora
`p->volts` — il valore del campo volts della
struttura `motor` puntata da `p`

`p->phases` — il numero di fasi della struttura
`motor` puntata `p`

Operazioni con struct

- Copia/assegnazione

```
struct motor p, q;  
p = q;
```

- Calcolo indirizzo

```
struct motor p;  
struct motor *s  
s = &p;
```

- Utilizzo dei campi

```
p.volts;  
s -> amps;
```

Operazioni con struct

- Si ricordi:
 - Passando una variabile struct per valore ha come effetto una *copia* nel sottoprogramma
 - Passando una struttura come variabile di ritorno al programma chiamante equivale anch'esso a una *copia* o *assegnazione*.

- Esempio:

```
struct motor f(struct motor g) {  
    struct motor h = g;  
    ...;  
    return h;  
}
```

Inizializzazione di una struttura

```
struct motor {  
    float volts;  
    float amps;  
    int phases;  
    float rpm;  
  
};           //struct motor
```

```
struct motor m = {208, 20, 3, 1800};
```

inizializza la struct

Attenzione alle dichiarazioni

- La seguente sintassi non è corretta:-

```
struct motor {  
    float volts;  
    float amps;  
    float rpm;  
    unsigned int phases;  
}; //struct motor
```

```
motor m;  
motor *p;
```

Occorre scrivere:

```
struct motor m;  
struct motor *p;
```

Typedef

- Definizione:– a **typedef** consente di *rinominare* un tipo
- Esempio,

```
typedef struct motor Motor;
```

```
Motor m, n;
```

```
Motor *p, r[25];
```

```
Motor function(const Motor m; ...);
```

typedef, consente di omettere “**struct**”

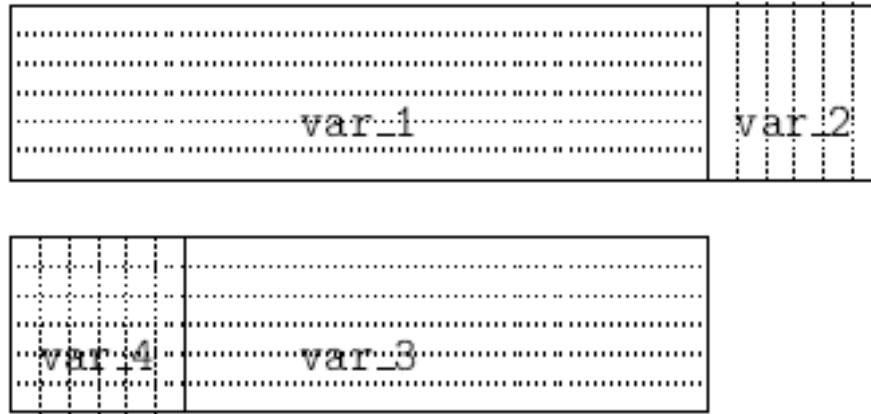
Unions

- Una **union** è come una **struct**, ma memorizza ***uno solo*** dei campi definiti ***alla volta***:
 - Una variabile union può contenere diversi tipi allo stesso tempo.
 - Lo spazio allocato deve essere tale da poter memorizzare il tipo più grande
 - I campi si sovrappongono

- Esempio,

```
union {  
    int ival;  
    float fval;  
    char *sval;  
} u;
```

Unions



Si noti che i due campi della variabile var union sono sovrapposti tra loro, e può essere utilizzato solo un campo alla volta.

Inoltre, non c'è modo di sapere se la var union contiene, in un dato istante, un intero oppure un carattere.

Unions

- E' il programmatore che deve tenere traccia del tipo volta per volta memorizzato!
- Esempio:

```
struct taggedItem {  
    enum {iType, fType, cType} tag;  
    union {  
        int ival;  
        float fval;  
        char *sval;  
    } u;  
};
```

Unions

```
struct taggedItem {  
    enum {iType, fType, cType} tag;  
    union {  
        int ival;  
        float fval;  
        char *sval;  
    } u;  
};
```

I campi della **struct** sono:–

enum tag;
union u;

Il valore di **tag** dice quale campo di **u** utilizzare

Unions

- **unions** sono utilizzate molto meno frequentemente delle **structs** —
- **Maggiormente:**
 - Nelle realizzazioni di dettaglio del sistema operativo
 - Nelle realizzazioni dei drivers
 - Nei sistemi embedded quando si ha accesso ai registri dell'hardware

Esercizi

- Definire una struct studente con campi nome, cognome, matricola, esami svolti, media voto
- Sviluppare un sottoprogramma che stampa una struct studente a video
- Sviluppare un sottoprogramma che legge una struct studente da tastiera
- Scrivere un sottoprogramma che legge un array di struct studente da tastiera
- Scrivere un sottoprogramma che cerca per matricola una struct studente in un array
- Scrivere un sottoprogramma che ordina un array per cognome
- Scrivere un sottoprogramma che stampa in un file binario la dimensione e un array di struct studente
- Scrivere un sottoprogramma che legge da un file binario la dimensione e un array di struct studente

Esercizi

- Definire una struct studente con campi nome, cognome, matricola, esami svolti, media voto
- Sviluppare un sottoprogramma che stampa una struct studente a video
- Sviluppare un sottoprogramma che legge una struct studente da tastiera
- Scrivere un sottoprogramma che legge un array di struct studente da tastiera
- Scrivere un sottoprogramma che cerca per matricola una struct studente in un array
- Scrivere un sottoprogramma che ordina un array per cognome
- Scrivere un sottoprogramma che stampa in un file binario la dimensione e un array di struct studente
- Scrivere un sottoprogramma che legge da un file binario la dimensione e un array di struct studente