

Sistemi Operativi: Modulo Laboratorio a.a. 14/15 Corso-A

struct & union in C

Le strutture (struct)

- Le strutture sono collezioni di una o più variabili che possono avere tipi diversi.
- Esempio:

```
struct studente {  
    char nome[64];  
    char cognome[64];  
    struct altreinfo {  
        char *via;  
        int    nc;  
    } info;  
    int matricola;  
};
```

Dichiarazione:

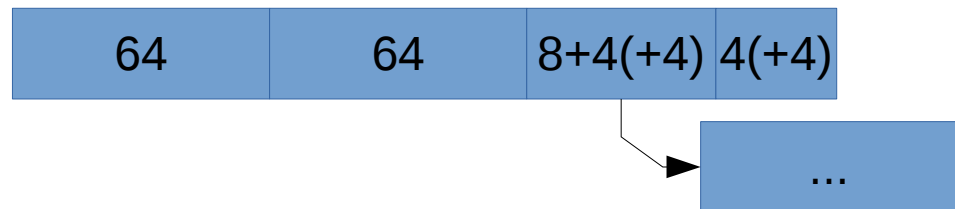
struct studente S; NOTA: '*struct studente*' è il tipo

Utilizzo:

```
S.matricola = 12345;  
printf("%s %s abita in via: %s n. %d\n",  
        S.nome, S.cognome, S.info.via, S.info.nc);
```

Rappresentazione di S in memoria:

supponiamo `sizeof(int) = 4`, `sizeof(char *) = 8`



(+4) è il padding per l'allineamento a 64-bit dato che gli indirizzi sono a 64-bit (`sizeof(char*)`)

NOTA: le strutture sono allineate di default a 4 o 8 bytes, per altri allineamenti va fatto esplicitamente

Operazioni con le struct

- Oltre alla manipolazione dei campi della struttura è possibile:

- Inizializzazione dei campi con valori costanti:

```
struct studente S1 = { "Mario", "Rossi", {"Roma", 1}, 1234};
```

- Copia ed assegnamento dell'intera struttura

```
void foo(struct studente S) { ....};          struct studente S2;  
foo(S1);                                     S2 = S1;
```

- Due strutture non possono essere confrontate!

- E' necessario confrontare i singoli campi con una funzione di compare specifica

```
If (S1 == S2) { ....} // Errore a compilazione
```

Bit-fields in strutture C

- Ad ogni campo di tipo `[unsigned]int` di una struttura è possibile specificare la sua ampiezza in bits
 - Alcuni compilatori come gcc supportano anche `[unsigned]short` ed `[unsigned]long`
- Sono usati per la gestione di maschere ed operazioni su bits.
- **ATTENZIONE** all'endianess della macchina !!! Codice che utilizza bit-fields è non portabile su differenti architetture

```
struct fields {
    unsigned int color    :3;
    unsigned int style    :2;
    int             :3; // padding
    char cell;
};
```

Dichiarazione:

```
struct fields F;
```

Utilizzo:

```
F.color = 3; F.style = 2; F.cell = 0;
```

```
printf("%d\n",*(int*)&F); //che valore stampa?
```

Union

- Una variabile di tipo *union* è una variabile che può contenere ad istanti diversi oggetti di tipo e dimensione diversa e che condividono tutti lo stesso spazio di memoria

```
union myUnion {
    int    val1;
    double val2;
    struct myStruct {
        char *p1;
        int    k:4;
    } val3;
};
```

Dichiarazione:

```
union myUnion U;
```

Utilizzo:

```
S.val1 = 12345;
printf("val2=%g\n", S.val2);
```

Rappresentazione di U in memoria:

supponiamo sizeof(char *) = 8

16

NOTA:

Le union sono utilizzate spesso per l'allineamento di strutture.

E' in grado di contenere val1, val2, e val3.

NOTA: le strutture e le union sono allineate di default a 4 o 8 bytes

Operazioni con le union

- Oltre alla manipolazione dei campi è possibile:

- Inizializzazione della union solo attraverso il primo campo:

```
struct myUnion U = {1234};
```

- Copia ed assegnamento dell'intera union

```
void foo(union myUnion U) { ....};          union myUnion U2;  
foo(U);                                     U2 = U;
```

- Due union non possono essere confrontate

```
If (U1 == U2) { ....} // Errore a compilazione
```