Le strutture in C

Prof. Francesco Accarino
IIS Altiero Spinelli via Leopardi 132
Sesto San Giovanni

I dati strutturati o definiti dall'utente

Nel linguaggio C è possibile definire tipi di dati detti strutturati attraverso la parola chiave struct aggregando più dati anche se di tipo diverso in un unico contenitore.

Una Struct:

- È formata da una o più variabili
- Contiene variabili di tipi diversi
- Riunisce più informazioni correlate
- Aiuta ad organizzare dati complessi

Le strutture

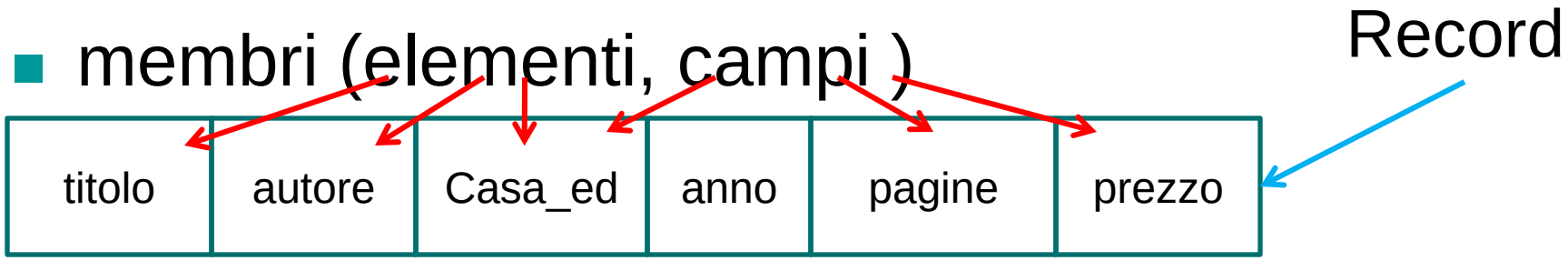
- Se ad esempio volessimo raccogliere tutte le informazioni riguardanti un libro potremmo scrivere:

```
struct libro {
char titolo[100];
char autore[50];
char casa_ed[100];
int anno, pagine;
double prezzo;
};
```

Le strutture

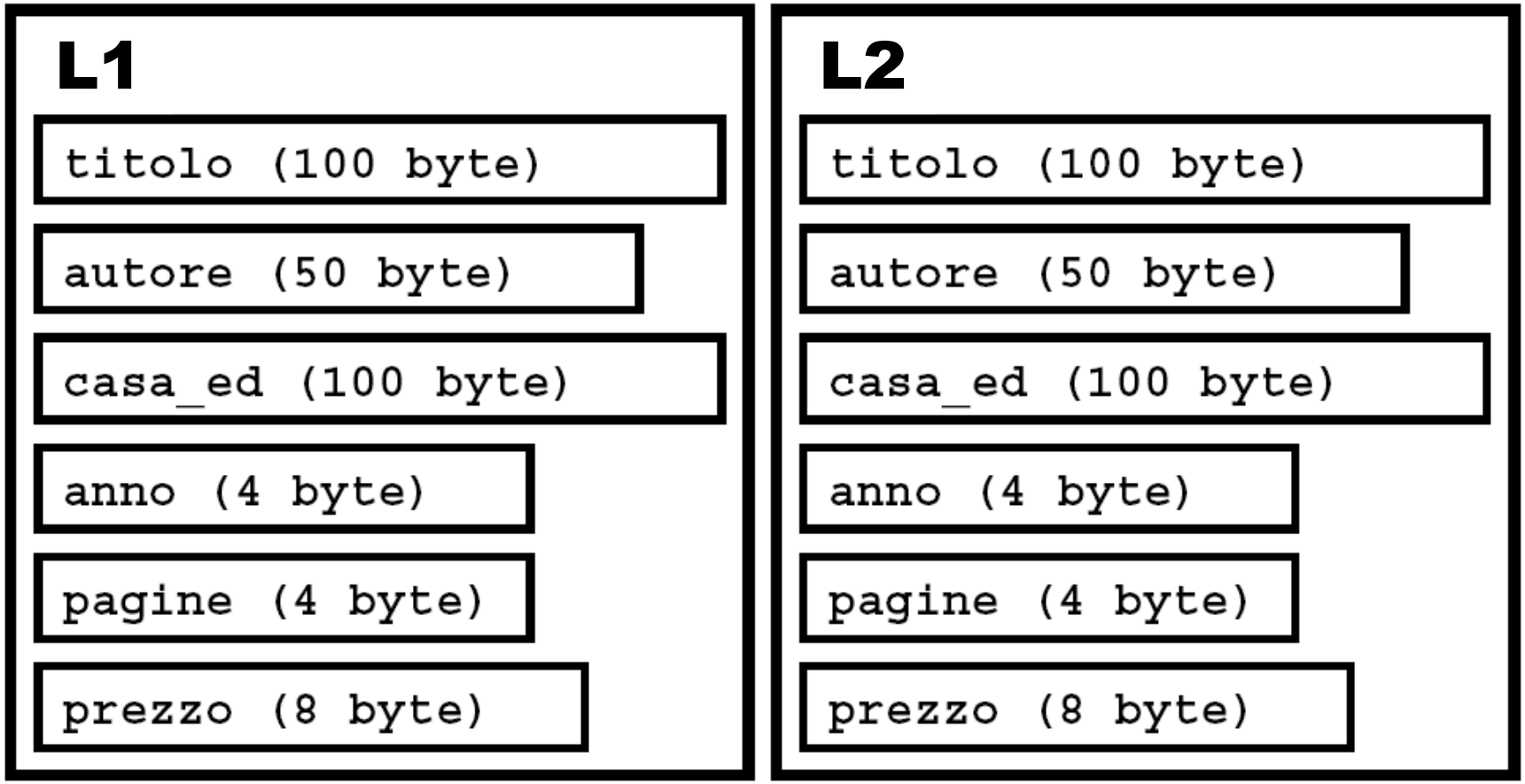
Con il codice scritto precedentemente:

- non è ancora creata nessuna variabile
- è stato solo definito un tipo di dato aggregato
- per dichiarare variabili di questo tipo:
- **struct libro L1,L2;**
- L1 e L2 sono istanze con la forma libro
- compilatore alloca memoria per tutti i
- membri (elementi, campi)



Le strutture

L1 ed **L2** rappresentano il riferimento alle celle di memoria dove sono contenute le in formazioni delle strutture memorizzate come sequenze di byte



Le strutture

- si possono dichiarare istanze anche cosi:

```
struct libro {
char titolo[100];
char autore[50];
char casa_ed[100];
int anno, pagine;
double prezzo;
} L1,L2;
```

Forma generale

La sintassi per dichiarare una struttura è molto semplice: bisogna iniziare con la parola chiave
struct seguita dal nome da noi scelto per questa struttura

```
struct nome_str {
tipo nome_campo,...,nome_campo;
tipo nome_campo,...,nome_campo;
...
} nome_var,..,nome_var;
```

e poi tra parentesi graffe l'elenco delle variabili o di sottostrutture che la compongono

Inizializzazione dei campi

I campi di una struct possono essere inizializzati all'atto della dichiarazioni delle istanze, ovviamente bisogna assegnare i valori rispettando ordine e tipi di dati

```
struct libro l1,l2={
"I Malavoglia",
"Giovanni Verga",
"Einaudi",
2001,
169,
9.8};
```

In questo esempio stiamo facendo 2 istanze della struct libro di nome **l1** ed **l2**. La seconda istanza è inizializzata con i valori riportati

Accedere agli elementi

- per accedere agli elementi di una strut si si usa la cosiddetta notazione puntata ovvero mediante l'operatore ".":

  nome_variabile.nome_campo

- si può usare in qualsiasi contesto

  come se fosse una semplice variabile:

```
printf("Titolo: %s",l1.titolo );
gets(l1.titolo);
```

Accedere agli elementi

- per accedere agli elementi si usa l'operatore ".":

```
strcpy(l1.titolo,"Le ore");
strcpy(l1.autore,"M. Cunningham");
strcpy(l1.casa_ed,"Bompiani");
l1.anno=2007;
l1.pagine=149;
l1.prezzo=6.8;
```

Accedere agli elementi

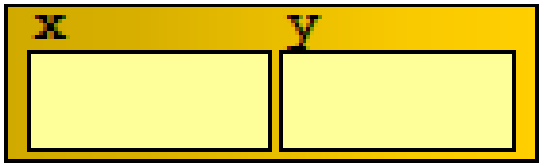
- per accedere agli elementi si usa l'operatore ".":

```
printf("%s\n",l1.titolo);
printf("%s\n",l1.autore);
printf("%s\n",l1.casa_ed);
printf("Anno: %d\n",l1.anno);
printf("Numero di pagine: %d\n", l1.pagine);
printf("Pr.: %.2f E\n",l1.prezzo);
```

Strutture innestate

Come già annunciato precedentemente tra gli elementi di una struttura può essere contenuta un'altra struttura basta che questa sia stata dichiarata prima

```
struct point {
   double x,y;
   };
struct triangolo {
   struct point p1,p2,p3;
   };
struct triangolo tr1,tr2;
```



Strutture innestate

- L’ accesso alle componenti della sottostruttura avviene ancora mediante notazione puntata. Avremo tanti punti per quanti sono i livelli di innesti.

Per esempio:

- per accedere alla coordinata **x** del secondo angolo del **tr1:**

  tr1.p2.x

Strutture innestate

- inizializzare la struttura innestata:

struct triangolo tr1={1.1,1.2,2.1,2.2,3.1,3.2};

- oppure in modo piu` leggibile:

```
struct triangolo tr1=
{  {1.1,1.2},
   {2.1,2.2},
   {3.1,3.2} };
```

Per le inizializzazioni di sottostrutture conviene usare le parentesi graffe (un pò come si faceva con le matrici) per rendere più leggibile il codice e non commettere errori

Vettori di strutture

La cosa veramente interessante di questi nuovi tipi di dati è che avendo dichiarato le caratteristiche di un libro se poi ho 1000 libri basta semplicemente dichiarare un vettore del tipo libro con numerosità 1000

- un vettore di strutture:

```
struct libro {
char titolo[100];
char autore[50];
char casa_ed[100];
int anno, pagine;
double prezzo;
} libri[1000];
```

E poi accedere ad ogni libro con la stessa sintassi dei vettori cioè mediante un indice, aggiungendo la notazione puntata. Ad esempio:

Libri[20].titolo

Rappresenta il titolo del libro di indice 20

Vettori di strutture

E così come per i vettori con un semplice ciclo for possiamo, stampare ad esempio, le in formazioni che ci interessano

- Si accede ai campi con la notazione puntata:

```
for(i1=0;i1<1000;i1++)
{
printf("%s\n",libr[i1].titolo);
printf("%s\n",libr[i1].autore);
printf("Prezzo: %.2f Euro\n",
libr[i1].prezzo);
}
```

Vettori di strutture

Le regole di prima valgono ovviamente anche per le strutture innestate

- se la struttura contiene un vettore:

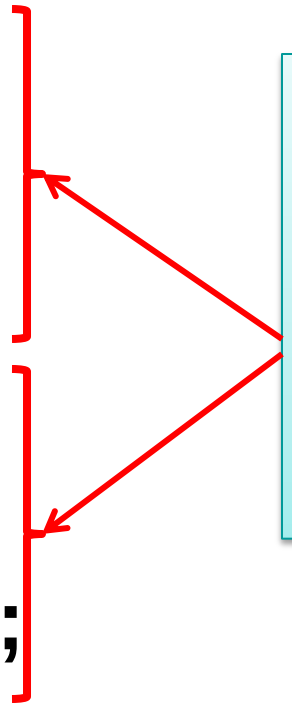
```
struct point {
double x[2];
};
struct triangolo {
struct point p[3];
};
struct triangolo tr[N];
```

Stiamo dichiarando un vettore di triangoli dove ogni triangolo contiene un vettore di punti che sono i vertici

Vettori di strutture

- per inizializzare il vettore:

```
struct triangolo tr[ ]={
{1.0,2.0,
  3.0,4.0,
   5.0,2.0},
{1.0,3.0,
   1.0,5.0,
   3.0,5.0} };
```



Con il codice qui di fianco stiamo dichiarando ed inizializzando un vettore contenete due triangoli e ognuno di essi contiene tre punti che sono i suoi vertici

Vettori di strutture

```
for(i1=0;i1<N;i1++)  //Per ogni triangolo i1
{
    printf("Triangolo n. %d:\n",i1);
    for(i2=0;i2<3;i2++)  //Per ogni vertice i2
    printf(" (%.2f,%.2f)\n", tr[i1].p[i2].x[0],
    tr[i1].p[i2].x[1]);
    }
```

Come si vede dal codice precedente ancora una volta mediante la notazione puntata e 2 cicli for possiamo stampare per tutti i triangoli i tre vertici che lirappresentano

Vettori di strutture

- output:

Triangolo n. 0:
(1.00,2.00)
(3.00,4.00)
(5.00,2.00)
Triangolo n. 1:
(1.00,3.00)
(1.00,5.00)
(3.00,5.00)

Di fianco è riportato il risultato di stampa del codice precedente dove si vede come per ogni triangolo vengono stampati i valori dei suoi vertici

Gli assegnamenti di strutture

Un aspetto interessante delle strutture è che è permesso l'assegnamento. In questo modo tutti i membri possono essere copiati usando un'unico assegnamento:

```
struct libro l1,l2;
strcpy(l1.titolo,"Le ore");
strcpy(l1.autore,"MichaelCunningham");
strcpy(l1.casa_ed,"Bompiani");
l1.anno=2001;
l1.pagine=169;
l1.prezzo=6.80;
l2=l1; //l2 conterrà la copia di tutti i campi di l1
```

Gli assegnamenti di strutture

- anche i vettori vengono copiati

```
struct vettore {
double v[MAX];
int lunghezza;
} v1;
struct vettore v2= {{1.0,2.2,4.5}, 3};
```

Gli assegnamenti di strutture

- l'assegnamento

  v1=v2;

  è consentito

- l'assegnamento

  v1.v=v2.v;

  non è consentito

Infatti in questo caso vengono automaticamente copiati tutti i campi di v2 in v1 anche se come campi vi fossero dei vettori o delle stringhe

In questo caso invece non è possibile copiare tutti gli elementi del secondo vettore nel primo mediante semplice assegnamento. Per ottenerlo dovremmo fare un ciclo di copia di ogni singolo valore

Strutture e funzioni

- tre modi di passare elementi di strutture:
  - –passare separatamente un membro
  - –passare un'intera struttura
  - –passare un puntatore ad una struttura

Strutture e funzioni

- si passa un membro solo quando non

c'è bisogno di tutta la struttura:

```
double euro_to_dollaro(double e)
{
return 1.49*e;
}
…
printf(" Prezzo: %.2f $",
   euro_to_dollaro(l2.prezzo));
```

Strutture e funzioni

- passare l' indirizzo di un membro

```
printf("Titolo: ");
gets(l1.titolo);
…
printf("Anno: ");
scanf("%d", &l1.anno);
```

Strutture e funzioni

- passaggio di un'intera struttura:chiamata per valore

In questo caso viene creata nella funzione mediante parametro formale una copia di tutti i campi della struttura rendendo così molto pesante il passaggio e ovviamente le modifiche alla copia non ricadranno sull'originale

```
void stampa(struct libro l)
    {
    printf("%s\n",l.titolo);
    printf("%s\n",l.autore);
    printf("Anno: %d\n",l.anno);
    printf("Prezzo: %.2f Euro\n",
    l.prezzo);
    }
```

La variabile l crea una copia di tutti i membri della struct

Strutture e funzioni

- la dichiarazione della struttura **libro**

deve essere globale

- devono corrispondere i nomi:

```
struct libro l1;
...
stampa(l1);
```

- non si può chiamare **stampa** con una

struttura uguale alla struttura **libro**

ma di nome diverso

Strutture e funzioni

- quando la struttura da passare è grande conviene usare puntatori
- puntatori alle strutture sono analoghi a quelli alle variabili ordinarie:

struct libro *lv;

- se lv punta ad una struttura

–*lv è la struttura

–(*lv).titolo, …, (*lv).prezzo sono i membri

Strutture e funzioni

- Con l'introduzione dei puntatori alle strutture viene introdotta la notazione freccia per semplificare l'accesso ai campi.
- se p è un puntatore ad una struttura

  p->nome_membro

  riferisce al membro nominato
- Senza questa abbreviazione avremmo dovuto scrivere

  (*p).nome_membro

Strutture e funzioni

- **void stampa(struct libro *l)**

```
{
printf("%s\n",l->titolo);
printf("%s\n",l->autore);
printf("Anno: %d\n",l->anno);
printf("Prezzo: %.2f Euro\n",l->prezzo);
}
```

In questo caso l contiene l'indirizzo della struttura passata e questo oltre a velocizzare il passaggio dei dati permette anche di modificare i dati originali

Strutture e funzioni

Ovviamente una funzione può anche ritornare una struttura sia come valore che come indirizzo

```
struct point {
double x,y;
};
struct point somma(struct point p1,struct point p2)
{
struct point s;
s.x=p1.x+p2.x;
s.y=p1.y+p2.y;
return s;
}
```

In questo caso viene ritornato una struttura per valore e cioè una copia di tutti i campi

Confronto di strutture

- le strutture non possono essere confrontate semplicemente con **==**
- si deveno realizzare funzioni che confrontano le strutture
- si possono definire uguaglianze di diversi tipi

Confronto di strutture

Questa funzione ritorna uno se i due libri hanno tutti i campi uguali zero altrimenti

```
int Confronta(struct libro *l1, struct libro *l2)
{
   if(strcmp(l1->titolo,l2->titolo)) return 0;
   if(strcmp(l1->autore,l2->autore)) return 0;
   if(strcmp(l1->casa_ed, l2->casa_ed)) return 0;
   if(l1->anno!=l2->anno) return 0;
   if(l1->pagine!=l2->pagine) return 0;
   if(l1->prezzo!=l2->prezzo)return 0;
   return 1;
}
```

Si ricorda che la funzione strcmp ritorna 0 se le due stringhe sono uguali 1 se la prima è maggiore -1 se la prima è minore

Puntatori alle strutture

- per allocare memoria per interi:

  int *v;

  v=(int *)malloc(10*sizeof(int));
- alloca memoria per 10 numeri interi:
- accedere a questi numeri:

  ***v, *(v+1), …, *(v+9)**

  oppure

  v[0],v[1],…,v[9]

Puntatori alle strutture

- si usa sizeof per sapere di quanta memoria c'è bisogno per la struttura:

  printf("Un libro occupa %d byte",
  sizeof(struct libro));

- output:

  Un libro occupa 264 byte

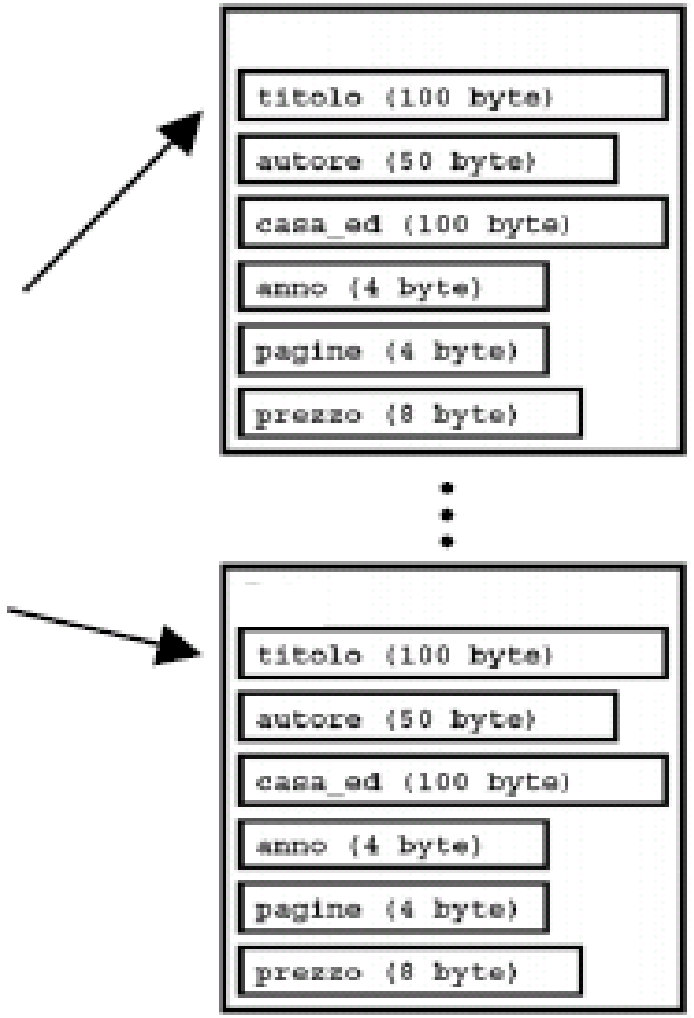
Puntatori alle strutture

```
struct libro *libri;
int n;

printf("Numero dei libri: ");

scanf("%d", &n);

libri=(struct libro*)malloc(sizeof(struct libro)*n);
```

Puntatori alle strutture

- n=5:

libri	025D48C0
libri+1	025D49C8
...	
libri+n-1	025D4CE0



Puntatori alle strutture

- per accedere al titolo dell'elemento numero 3:

(*(libri+2)).titolo

oppure

libri[2].titolo

oppure

(libri+2)->titolo

Le unioni

Le union servono per condividere un indirizzo di memoria tra due o più variabili di tipi e dimensioni diversi. La forma generale è la seguente:

```
union nome{
tipo nome_variabile;
tipo nome_variabile;
…
} variabili_unione;
```

Le unioni

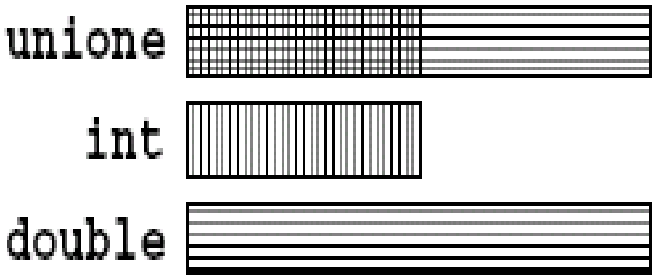
La dichiarazione del tipo union e la conseguente dichiarazione delle variabili è del tutto simile alle strutture. Cambia profondamente il risultato ottenuto. Infatti per le union non si hanno tanti campi quante sono le variabili dichiarate, ma semplicemente un'unica variabile che può assumere le forme diverse dichiarate e la memoria allocata corrisponde alla variabile massima

- per esempio:

```
union u {
  int i;
  double d;
};
union u u1;
```

- oppure

```
union u {
  int i;
  double d;
} u1;
```



L' unione occupa 8 byte
- **u1.i** riferisce ai primi 4 byte
- **u1.d** riferisce a tutti i byte

Le enumerazioni

Le enumerazioni sono una lista di valori interi, in pratica una sequenza di costanti che rende più leggibile il codice

- forma generale:

  enum nome { elenco } variabili;

- per esempio:

  enum boolean {FALSE,TRUE};

- il primo nome ha valore 0, il secondo 1, e cosi via, a meno che non è specificato diversamente

Le enumerazioni

```
enum stile {GIALLO=1,ORRORE,POESIA,DRAMMA};
struct libro_e
{
char titolo[100];
char autore[50];
enum stile st;
float prezzo;
};
```

In questo esempio è come se avessimo definito quattro costanti di nome **GIALLO ,ORRORE,POESIA,DRAMMA con valori 1,2,3,4.** Seviene specificato ilvalore del primo elememto gli altri sono conseguenti

Definire nuovi tipi

il C consente di definire nuovi tipi mediante la parola chiave typedef.

Esempio:

typedef int Lunghezza;

- crea un sinonimo di int:

Lunghezza l1,l2;

Lunghezza l3[10];

Definire nuovi tipi

Questa possibilità è utile soprattutto per le strutture infatti definendo come nuovo tipo di dato una struttura si semplifica la dichiarazione delle variabili di tipo struttura

Esempio:

```
struct libro
{
char titolo[100];
char autore[50];
};
typedef struct libro Libro;
```

da adesso posso scrivere **Libro** ← Libro nuovo tipo di dato

invece di **struct libro**

Libro l1;