UNIVERSITÀ
degli STUDI
di CATANIA

Strutture, enum, union

Corso di programmazione I (A-E / O-Z) AA 2023/24

Corso di Laurea Triennale in Informatica

Fabrizio Messina
fabrizio.messina@unict.it

Dipartimento di Matematica e Informatica

Strutture (struct)

Le strutture sono molto usate nel linguaggio C.

Definite mediante la keyword **struct**.

Una struttura è un tipo **composto di un insieme di dati** (comunemente detto record).

La struttura C è l'antenato della classe C++.

```
1   struct record{
2     char *nome;
3     char *cognome;
4     float saldo;
5   } s1, s2;
```

Definisce il tipo struct record e variabili s1 ed s2 dello stesso tipo.

Strutture (struct)

Inizializzazione di strutture.

Può avvenire 1) con una lista di inizializzazione ed eventuale uso di "designatori".

```
1 struct struttura s1 = {}; // tutto a zero
2 struct struttura s2 = {"Mario", "Rossi", 123.45};
3 struct struttura s2 = {.cognome="Rossi", .nome="Mario", \
4 .saldo=123.45};
5 //..
```

2) oppure dopo la definizione, con l'operatore "punto" '.'

```
1 s1.nome = "Elena";
2 s1.cognome = "Bianchi";
3 s1.saldo = 345.67;
4 //..
```

Strutture (struct)

Esempi svolti

`19_01.c`

Strutture (struct)

Puntatori a strutture, strutture autorefenziali

La struttura non può contenere una variabile dello stesso tipo della struttura, ma può **contenere un puntatore alla struttura stessa.**

```
1   struct record{
2     char *nome;
3     char *cognome;
4     float saldo;
5     struct record *next;
6   };
```

NB: Puntatore a struttura dello stesso tipo utile a creare **strutture dati collegate!**

Confronto di strutture

Vanno confrontati i membri, singolarmente!

Non è possibile confrontare due variabili dello stesso tipo struct con gli operatori "==" e "!=", in quanto alcuni campi della struttura potrebbero essere allineati ad una word di memoria pur essendo più piccoli della word stessa.

Esempio.

```
1 struct record{
2 char c; // 1 byte, ma (probabilmente) allineato a 4 byte ..
3 int k; // 4 byte!
4 };
```

Strutture (struct)

Il carattere potrebbe essere contenuto in una word di 4 byte, anche se viene usato solo un byte!

char	??	??	??	int			
1 b	1 b	1 b	1 b	1 b	1 b	1 b	1b

1b = 1 byte

Il confronto opererebbe sui rimanenti 3 byte che conterrebbero bit a casaccio!

Vedi esempi svolti (`19_02.c`)

Strutture (struct)

Esempi svolti

`19_02.c`

Strutture (struct)

Puntatori a strutture, accesso ai dati, operatore "narrow" $(->)$.

```
 1 struct record {
 2     float x;
 3     float y;
 4     // ..
 5 };
 6
 7 struct record *rec =
 8     (struct record *) malloc(sizeof(struct record));
 9 rec->x = 0.3;
10 rec->y = 0.7;
```

Strutture (struct)

Puntatori a funzioni e strutture.

Consentono di associare (debolmente) le funzioni ai dati.

```
1   struct record{
2     float x, y;
3     float (*sum) (struct record *ptr);
4 };
5
6   float sum_func(struct struttura *ptr){
7     return ptr-> x + ptr->y;
8     }
9
10  struct record rec;
11  rec.x = rec.y = 0.9;
12  rec.sum = sum_func; // associa la funzione al puntatore
13  float sum = rec.sum(&rec);
```

Strutture (struct)

Puntatore a funzione vs funzione.

```
1   int sum(int a, int b);
2
3   int (*sum_ptr)(int a, int b);
```

Linea 1: **funzione di nome sum** con due paramatri formali int che restituisce un intero.

Linea 3: **puntatore ad una funzione** che prende due paramatri formali int e che restituisce un intero. Il nome del puntatore è `sum_ptr`.

Strutture (struct)

Esempi svolti

`19_03.c`

Strutture (struct)

Uso della parola chiave **typedef**.

La parola `typedef` permette di creare *alias* di dati precedentemente definiti (anche tipi primitivi).

```
1     typedef struct record {
2         float x;
3         float y;
4         //..
5     } srec;
6     // ...
7     srec myrecord;
8     srec *ptr_record;
9     //...
```

Strutture (struct)

Esempi svolti

`19_04.c`

Enumerazioni (enum)

Una enumerazione è un tipo di dato che può assumere un valore tra un insieme di **nomi** definiti all'interno di essa.

```
1    enum Mese{
2        Gennaio,
3        Febbraio,
4        //...
5        Dicembre
6    };
7    //..
8    enum Mese m = Gennaio;
```

Enumerazioni (enum)

I **nomi** definiti in una enum sono "esportati" nello scope in cui la enumerazione è stata definita;

```
1    enum Mese{
2        Gennaio, Febbraio, /* ... */, Dicembre
3    };
4    //..
5    enum Mese m = Gennaio;   // OK
6    m = Dicembre; //OK
```

Istruzioni alle linee **5 e 6** fanno riferimento a due "nomi" definiti nella enumerazione.

Enumerazioni (enum)

I **nomi** di una enumerazione sono rappresentati in memoria come numeri interi, i cui valori, se non specificati dal programmatore, **partono da zero**.

Conversioni `int`→`enum` e conversioni `enum`→`int` sono entrambe valide.

```
1   enum Mese{
2       Gennaio, Febbraio, /* ... */ Dicembre // 0,1,2,..
3   };
4   //..
5   enum Mese m = Gennaio;   // OK
6   enum Mese m2 = 10; // OK, m2 = Novembre
7   int mese = m2; // enum-->int implicita, OK
```

Enumerazioni (enum)

Le enumerazioni si possono usare nei costrutti `switch`

```
1   const char* traduci_mese(enum Mese m){
2   switch(m){
3     case Gennaio:
4       return  "Gennaio";
5       break;
6     case Febbraio:
7       return "Febbraio";
8       break;
9     //...
10    case Dicembre:
11      retun "Dicembre";
12      break;
13  }
14 }
```

Enumerazioni (enum)

Scope vs name clash.

```
1   enum Mesi{
2       Gennaio,
3       Febbraio,
4       /* ... */,
5       Dicembre
6   };
7
8   //Compile-time error! name clash
9   enum MeseEstivo{
10      Giugno, Luglio, Agosto
11  };
```

Enumerazioni (enum)

Scope vs rappresentazione delle enumerazioni.

```
1   enum MeseInvernale{//0,1,2
2     Dicembre, Gennaio, Febbraio
3   };
4
5   enum MeseEstivo{//0,1,2
6     Giugno, Luglio, Agosto
7   };
8   //..
9   enum MeseInvernale m = Dicembre; //Dicembre=0, Giugno=0
10  if(m==Giugno) //Attenzione! Per default nessun warning..
11    //...
```

Enumerazioni (enum)

Valori corrispondenti alle enumerazioni.

Per default partono da zero, ma il programmatore può specificarli.

Dopo ogni valore specificato per un simbolo, il successivo (se non specificato dal programmatore), viene assegnato sulla base del valore precedente..

```
1   enum Mese{
2       Gennaio, // 0
3       Febbraio = 20,
4       Marzo, // 21
5       Aprile, // 22
6       Maggio = 5,
7       Giugno, // 6
8       // ...
9   };
```

Enumerazioni (enum)

Esempi svolti

`19_05.c`

Union

Una **union** è una struttura i cui membri sono allocati tutti a partire dallo stesso indirizzo di memoria.

Lo spazio realmente occupato in memoria sarà uguale alla dimensione del campo più grande.

Di conseguenza tale struttura potrà contenere non più di un dato alla volta.

```
1   union Value {char c; int num;};
2   //...
3   union Value v;
4   v.c = 'z';
5   //...
6   v.num = 10;
```

Union

Il record può essere un carattere oppure un numero.

La struttura di tipo union consente di rsparmiare spazio in memoria.

```
1   enum Type { number, character };
2
3   union Value{ int num; char c; };
4
5   struct Record { enum Type t; union Value v };
```

Union

Esempi svolti

`19_06.c`

Riferimenti e approfondimenti

[1] → Capitolo 10 (fino al 10.6).
[1] → Capitolo 10.8, 10.11. [1] → Test di autovalutazione presenti alla fine del Capitolo 10, solo su argomenti trattati.

---

[1] Paul J. Deitel and Harvey M. Deitel.
C Fondamenti e tecniche di programmazione.
Pearson, 2022.

HOMEWORK

Homework H19.1

Definire una struttura (struct) `record` che contenga un numero in virgola mobile, un carattere e due puntatori a caratteri `S` ed `W` (due stringhe).

Codificare una funzione che prenda in input un puntatore ad una tale struttura e che sia in grado di inizializzare la struttura con elementi pseudo-casuali. In particolare il campo W dovrà essere inizializzato con caratteri appartenenti all'insieme [0-9], il campo S con caratteri in nell'insieme [a-z].

Invocare opportunamente la funzione all'interno della funzione `main`.

Homework H19.2

Estendere l'esercizio del punto precedente con la definizione di una struttura `data` che contenga un intero chiamato `ID`, ed un campo di tipo `record` definito nel precedente esercizio (struttura annidata). ID andrà inizializzato sempre con numeri positivi.

All'interno della funzione `main()`, creare un un array di N elementi di tipo `data`, ed inizializzare un numero $k \leq N$ di elementi di tale array, con k a piacere. L'elemento successivo al k-esimo elemento dello array (se $k < N$), ovvero il primo elemento "vuoto" sarà distinguibuile dagli altri perchè $ID == -1$.

Codificare una funzione di inizializzazione che prenda in input l'indirizzo di un elemento `data`, che faccia uso della funzione definita nel precedente esercizio, e che sia in grado di assegnare ID **univoci** ad ogni elemento `data`.

Homework H19.3

Estendere il precedente esercizio con la modifica del tipo `data` in modo che includa anche un campo `FLAG` che prenda valori definiti in una enumerazione (assegnare alla enumerazione un nome a piacere): la enumerazione avrà possibili valori `VOWELS` e `THREE`.

Codificare una opportuna funzione che prenda in input l'array di elementi di tipo `data` per modificare ognuno di questi nel seguente modo: se il numero di vocali presenti nel campo S è maggiore del numero di elementi divisibili per tre del campo W, allore `FLAG=VOWELS`, altrimenti `FLAG=THREE`.

Homework H19.4

Estendere il precedente esercizio con la modifica del tipo `data` in modo che includa anche un campo di tipo `union` di nome `FLAG_DATA`. La union (assegnare al tipo union un nome a piacere) sarà composta di un campo `unsigned short` ed un campo `char`.

Codificare una opportuna funzione che prenda in input l'array di elementi di tipo `data` per modificare ognuno di questi nel seguente modo: se `FLAG=VOWELS`, allora pone in `FLAG_DATA` la prima vocale presente nel campo S, altrimenti se `FLAG=THREE` pone in `FLAG_DATA` il primo numero divisibile per tre presente nel campo S.

FINE