Strutture, Unioni, Enumerazioni in C

Emilio Di Giacomo

Strutture

- Una struttura è una collezione di variabili raggruppate sotto uno stesso nome
  - da questo punto di vista è simile a un array
- Le variabili che compongono una struttura possono essere di tipo diverso
  - negli array invece le variabili sono necessariamente dello stesso tipo
- Le strutture sono utilizzate per rappresentare dati compositi, cioè costituiti da più attributi elementari
- Usate insieme ai puntatori permetto di realizzare strutture dati dinamiche

Strutture

- Le strutture sono tipi di dati derivati, cioè definiti a partire da altri tipi di dato
- Le variabili che costituiscono una struttura sono detti membri (o campi) della struttura
- La sintassi per definire una struttura è la seguente

```
struct <etichetta> {
        <definizione dei membri>
};
```

- La definizione di ogni membro è una definizione di variabile
  - va indicato il tipo e il nome della variabile

Strutture: esempio

- Supponiamo di voler scrivere un programma che realizza un gioco di carte
- Avremmo bisogno di rappresentare le diverse carte da gioco
- Ogni carta da gioco ha due attributi:
  - il valore (Asso, Due, Tre, ..., Fante, Cavallo, Re)
  - il seme (Coppe, Denari, Bastoni, Spade)
- Possiamo definire il tipo *carta* tramite una struttura

Strutture: esempio

```
struct carta {
        char * valore;
        char * seme;
};
```

- *carta* è l'etichetta (cioè il nome) della struttura
- *valore* e *seme* sono i membri della struttura
- la definizione di una struttura deve essere terminata da un punto e virgola

Strutture

- Nell'esempio precedente i due campi della struttura erano dello stesso tipo
- I campi possono essere di tipo diverso
- Possono essere sia di tipo primitivo (*int*, *float*,...) sia di tipi aggregati, cioè array o altre strutture
- Campi della stessa struttura devono avere nomi distinti
  - campi di strutture diverse invece possono avere lo stesso nome

Strutture: esempio

```
struct impiegato{
        char nome[20];
        char cognome[20];
        unsigned int eta;
        char sesso;
        double salario;
};
```

Definizione di variabili

- Una volta definita una struttura essa definisce un tipo
- Tale tipo va denotato come *struct <etichetta>* e può essere usato come qualunque altro tipo
- Esempio:

```
struct carta unaCarta;

struct carta mazzo[40];

struct carta * cPtr;
```

Definizione di variabili

- Variabili di tipo *struct* possono essere anche dichiarate contestualmente alla definizione della struttura
- Esempio:

```
struct carta {
        char * valore;
        char * seme;
} unaCarta, mazzo[40], * cPtr;
```

Etichette

- L'etichetta di una struttura può anche essere omessa
- In questo caso però le variabili possono essere dichiarate soltanto nella definizione della struttura
- Esempio:

```
struct {
        char * valore;
        char * seme;
} unaCarta, mazzo[40], * cPtr;
```

Strutture

- Una struttura non può contenere un campo il cui tipo sia quello della struttura stessa

```
struct impiegato{
        char nome[20];
        char cognome[20];
        unsigned int eta;
        char sesso;
        double salario;
        struct impiegato capo; // ERRORE
};
```

Strutture

- È però possibile avere un campo che sia un puntatore alla struttura stessa

```
struct impiegato{
        char nome[20];
        char cognome[20];
        unsigned int eta;
        char sesso;
        double salario;
        struct impiegato * capo; // OK
};
```

Strutture autoreferenziali

- Si parla in questo caso di strutture autoreferenziali
- Vedremo che l'uso di strutture autoreferenziali è centrale nella realizzazione di strutture dati dinamiche

Inizializzazione di strutture

- Le strutture possono essere inizializzate in maniera simile agli array
- Esempio:

struct carta unaCarta = {"Asso", "Spade"};

struct impiegato mRossi= {"Mario", "Rossi", 32, 'M', 1800.0}

- Se i valori nella lista sono meno dei campi, quelli mancanti vengono inizializzati a *0* (o *NULL*)

Accesso ai campi

- È possibile accedere (in lettura o scrittura) ai singoli campi di una struttura tramite due operatori
- L'operatore di membro di struttura (.), anche detto operatore punto, permette di accedere ad un campo della struttura indicandone il nome
- L'operatore di puntatore a struttura (->), anche detto operatore freccia, permette di accedere ad un campo di una struttura tramite un puntatore
  - è la combinazione dell'operatore di deferenziazione (*) e dell'operatore punto (.)

Accesso ai campi: esempi

- Operatore punto:

```
struct carta unaCarta={"Asso", "Coppe"};
printf("%s", unaCarta.valore); // stampa Asso
printf("%s", unaCarta.seme); // stampa Coppe
```

- Operatore freccia:

```
struct carta unaCarta={"Asso", "Coppe"};
struct carta * cPtr=&unaCarta;
printf("%s", cPtr->valore); // stampa Asso
printf("%s", cPtr->seme); // stampa Coppe
```

Accesso ai campi: esempi

- Le due espressioni seguenti sono equivalenti:

cPtr->valore

*(*cPtr).valore*

Strutture e parametri

- È possibile passare una struttura come parametro a una funzione
- La struttura viene passata per valore
  - viene quindi creata una copia di tutti i campi
  - la struttura non è modificabile dalla funzione
- Volendo passare una struttura per riferimento si deve
  - definire il parametro come puntatore al tipo struttura
  - all'atto dell'invocazione passare l'indirizzo tramite l'operatore di indirizzo

Strutture e parametri: esempi

```
void stampaCarta(struct carta c){
        printf("%s di %s", c.valore, c.seme);
}
void creaCarta(struct carta * c){
        c->valore="Asso";
        c->seme="Coppe";
}
...
struct carta unaCarta;

creaCarta(&unaCarta);
stampaCarta(unaCarta);
```

La parola chiave typedef

- La parola chiave *typedef* permette di definire un alias (cioè un sinonimo) per un tipo di dato
- Esempio:

  typedef int intero

- *typedef* non definisce un nuovo tipo ma solo un nuovo nome per indicare un tipo esistente
- *typedef* viene spesso usato con i tipi struttura per semplificare l'uso del tipo

La parola chiave typedef

```
struct carta {
        char * valore;
        char * seme;
};

struct carta unaCarta;
struct carta mazzo[40];
```

```
struct carta {
        char * valore;
        char * seme;
};
typdef struct carta Carta;

Carta unaCarta;
Carta mazzo[40];
```

La parola chiave typedef

- È possibile anche scrivere:

```
typedef struct {
        char * valore;
        char * seme;
} Carta;

Carta unaCarta;
Carta mazzo[40];
```

Ulteriori commenti sulle strutture

- È possibile assegnare una variabile di tipo struttura ad una variabile dello stesso tipo
  - Ciò non è possibile con gli array
  - Viene creata una copia della variabile assegnata
- Esempio:

  struct carta unaCarta={"Asso", "Coppe"};

  struct carta altraCarta=unaCarta;
- I confronti di uguaglianza e non uguaglianza tra strutture non si può fare con gli operatori == e *!=*
  - vanno confrontati i singoli campi

Unioni

- L'unione è un costrutto simile alla struttura
- Come la struttura un'unione ha diversi campi
- Ma un solo campo alla volta può essere utilizzato
- I diversi campi condividono la stessa porzione di memoria
- Non vedremo ulteriori dettagli sulle unioni

Enumerazioni

- Un'enumerazione è un tipo che viene definito elencando i suoi valori
- I tipi enumerativi vengono di solito usati quando si vuole rappresentare un tipo i cui valori appartengono ad un insieme di cardinalità limitata
- Esempi tipici sono i giorni della settimana, i mesi dell'anno o i semi delle carte da gioco

Enumerazioni: esempio

```
enum giorni {LUN, MAR, MER, GIO, VEN, SAB, DOM};
...
enum giorni g=MAR;
```

```
enum semi {COPPE,DENARI,BASTONI,SPADE };
struct carta{
        char *valore;
        enum semi seme;
};
struct carta unaCarta={"Asso",COPPE};
```

Enumerazioni

- I valori di una enumerazione sono costanti intere
- I valori interi corrispondenti alle costanti sono i valori da *0* a *n-1* (se *n* sono i valori)
- È possibile assegnare esplicitamente valori diversi

Enumerazioni: esempio

```
#include <stdio.h>

enum giorni {LUN, MAR, MER, GIO, VEN, SAB, DOM};

int main(){
      char * nomi[] = {"Lunedi", "Martedi",
            "Mercoledi", "Giovedi", "Venerdi",
            "Sabato", "Domenica"};

      enum giorni giorno;

      for(giorno=LUN; giorno<=DOM; giorno++)
            printf("%s\n", nomi[giorno]);
}
```