12.7 Tipi derivati composti tramite struttura

La struttura aggrega un insieme di membri che possono appartenere a qualsiasi tipo. Per tale motivo la struttura è un costrutto potente che consente di rappresentare come un tutt'uno oggetti di tipo e dimensione diversi. Per esempio, volendo rappresentare il concetto di anagrafe potremmo definire un insieme di strutture nel seguente modo:

```
struct data {
        int giorno;
        char mese [20];
```

```
   int anno;
};
struct ind {
   char via[35];
   int numero;
   char interno
   char citta[30];
   char prov[2];
};
struct persona {
   char nome[30];
   char cognome[30];
   struct data data_nasc;
   char comune_nasc[30];
   struct ind indirizzo;
   char telefono[10];
   char parentela[2];
};
```

Le strutture `data`, `ind` e `persona` hanno membri di tipo e dimensione differenti. In particolare si osserva che la struttura `persona` ha dei membri che sono a loro volta delle variabili struttura: `data_nasc` è di tipo `data` e `indirizzo` è di tipo `ind`. In C è possibile definire strutture di strutture purché le strutture che intervengono come membri siano state definite prima della struttura che le contiene.

Questo, ricordando le regole di visibilità di una variabile, è assolutamente normale. Ma cosa succede se una generica struttura `S` ha un membro che a sua volta è una struttura dello stesso tipo `S`? Se per esempio volessimo mantenere una lista di persone ordinata per ordine alfabetico, dovremmo prevedere nella struttura `persona` un membro di tipo persona che dice qual è la prossima persona nell'elenco ordinato. In C, pedrò, una struttura `S` non può contenere un membro di tipo puntatore a `S`. Così il problema della lista ordinata potrebbe essere risolto con:

```
struct persona {
      char nome[30];
      char cognome[30];
      struct data data_nasc;
      char comune_nasc[30];
      struct ind indirizzo;
      char telefono[10];
      char parentela[2];    /* CF capofamiglia, CG coniuge ecc. */
      struct persona *link; /* punta alla persona seguente */
     }
```

Il caso delle strutture ricorsive è molto più comune di quanto si possa pensare. Alcune delle più importanti strutture dati quali le liste, le pile e gli alberi sono rappresentabili mediante strutture ricorsive; a tale proposito il lettore troverà numerosi esempi in successivi capitoli di questo testo.

Se la struttura è lo strumento con cui si rappresenta una classe di oggetti (le automobili, gli impiegati, i nodi di un albero ecc.), aggregando un insieme di membri è logico aspettarsi una molteplicità di elementi, cioè di variabili. Tornando per esempio al caso dell'anagrafe, avrebbe poco senso concepire tale struttura per modellare una sola persona. Piuttosto è significativo rappresentare un insieme di persone, tutte di tipo `anagrafe`, e quindi adoperare il costrutto array:

```
struct persona anagrafe[300];
```

L'identificatore `anagrafe` è un array di 300 variabili di tipo `persona`, ovvero modella un'anagrafe con al più 300 elementi.

Le strutture e i tipi composti con le strutture sono molto importanti nella programmazione C, perché consentono di organizzare in modo razionale i dati del problema in esame. Un programma, però, non descrive solo dati ma anche funzioni. Per esempio, nel caso dell'anagrafe ci preoccuperemo di scrivere funzioni per l'inserimento di una persona, per la sua cancellazione, ricerca e visualizzazione; quindi è logico aspettarsi in un programma C funzioni che lavorano su variabili di tipo struttura. Per passare una variabile di tipo struttura a una funzione occorre far riferimento a un puntatore alla struttura.

Consideriamo a titolo di esempio il programma del Listato 12.2. La nostra anagrafe è rappresentata da un array di 30 elementi di tipo struttura, `anag`. La struttura dati che modella una persona è molto semplice e comprende:

- cognome
- nome
- indirizzo
- eta

Listato 12.2 Gestione anagrafica

Il programma, per mezzo della funzione men_per, presenta il seguente menu:

```
    ANAGRAFE

1. Immissione Persona
2. Cancellazione Persona
3. Ricerca Persona
4. Visualizza Anagrafe
0. Fine

Scegliere un'opzione:
```

Con la scelta 1, Immissione Persona, si lancia la funzione ins_per che inserisce i dati di una persona nell'array anag; la funzione è molto semplice: non effettua controllo sull'esistenza di una persona nell'array anag, ma inserisce dati finché lo consentono le dimensioni dell'array. Con la scelta 2, Cancellazione Persona, viene invocata la funzione can_per che richiede dapprima all'utente i seguenti dati:

```
CANCELLA PERSONA
     ---------------
     Cognome  :
     Nome  :
     Età  :
```

e poi invoca la funzione cer_per. Quest'ultima prende in ingresso le variabili cognome, nome ed età inserite dall'utente, effettua una ricerca sequenziale nell'array anag, e se trova la persona corrispondente ai dati forniti in ingresso restituisce il puntatore, ps, alla prima occorrenza dell'array che corrisponde ai dati richiesti dall'utente. Il lettore osservi come cer_per sia un classico esempio di funzione che restituisce un puntatore a variabile di tipo struttura:

```
struct per *cer_per(char *cg. char *nm, int et)
     {
     ...
     }
```

Se per cer_per non trova nessuna persona con i dati richiesti allora restituisce un puntatore NULL. Nel caso in cui invece cer_per trovi una occorrenza valida, la funzione can_per prende il puntatore all'occorrenza e lo passa a vis_per, che ha lo scopo di visualizzare i dati della persona che si vuol eliminare. La funzione vis_per è un classico esempio di funzione che prende in ingresso un puntatore a struttura:

```
void vis_per(struct per *p)
     {
     ...
     }
```

Solo dopo conferma da parte dell'utente la persona è eliminata da eli_per, altra funzione che prende come ingresso un puntatore a variabile struttura. Comportamento analogo a quello di can_per presenta la funzione ric_per, invocata scegliendo dal menu la voce 3, Ricerca Persona. L'ultima opzione del menu permette di visualizzare uno a uno i dati di tutte le persone presenti in anag.