Cristiana Bolchini

01.11.2015
Joint Special Section on the IEEE TC and TNano
Deadline extension for submission to a Joint Special Section on IEEE Trans. on Computers and IEEE Trans. on Nanotechnology [...]

L'array è un esempio di struttura dati. Utilizza dei tipi di dati semplici, come int, char o double e li organizza in un array lineare di elementi. L'array costituisce la soluzione in numerosi casi ma non tutti, in quanto c'è la restrizione che tutti i suoi elementi siano dello stesso tipo.In alcuni casi è però necessario poter gestire all'interno della propria struttura un mix di dati di tipo diverso. Si consideri a titolo d'esempio l'informazione relativa al nominativo, anni e salario. Il nominativo richiede una stringa, ossia un array di caratteri terminati dal carattere `'\0'`, l'età ed il salario richiedono interi.

Con le conoscenze acquisite fino ad ora è possibile solamente dichiarare delle variabili separate, soluzione non altrettanto efficiente dell'utilizzo di una unica struttura dati individuata da un unico nome: il Linguaggio C a tale scopo dispone della struct.

Definizione di una struct

La dichiarazione di una struct è un processo a due fasi. La prima è la definizione di una struttura con i campi dei tipi desiderati, utilizzata poi per definire tutte le variabili necessarie con la suddetta struttura. Si consideri il seguente esempio: si desidera gestire i dati di un insieme di persone relativamente al loro nominativo, all'età e al loro salario. Per prima cosa si definisce la struttura che consente di memorizzare questo tipo di informazioni, nel seguente modo:

```
struct s_dipendente
{
  char nominativo[40];
  int anni;
  int salario;
};
```

A questo punto è possibile definire una variabile con la struttura appena introdotta:

struct s_dipendente dipendente;

La variabile si chiama dipendente ed è del tipo `struct s_dipendente` definito precedentemente.

La sintassi per la definizione di una struttura è la seguente:

```
struct nome_struttura
{
  lista dei campi (tipo-nome)
};
```

In seguito, è possibile definire variabili come segue:

struct nome_struttura nome_variabile;

Accesso ai campi della struttura

Per accedere ai campi della struttura, per scrivere un valore o per leggerlo, è necessario indicare il nome della variabile seguito da quello del campo di interesse, separati da un punto .. Ad esempio, per la struttura precedentemente dichiarata:

```
struct s_dipendente dipendente
...
dipendente.anni = 30;
...
printf("%s\n", dipendente.nominativo);
...
```

Una volta individuato il campo d'interesse mediante la sequenza nome_variabile.nome_campo si ha a che fare con una variabile normale, e nel caso di `dipendente.anni` ad una variabile di tipo intero. Si noti che non deve utilizzare il nome della struttura `s_dipendente`, ma il nome della variabile, anche perchè ci possono essere più variabili con la stessa struttura. Nel caso in cui una struct contenga campi costituiti da altre struct si utilizzano i nomi di ogni struttura separati da punti fino a quando non si arriva al campo finale della struttura. Si consideri il seguente esempio:

```
struct s_azienda
{
  char nome[30];
  ...
  struct s_dipendente contabile;
  ...
};
```

Per accedere ai dati del contabile si segue il percorso: azienda.contabile.nominativo.

Per riassumere, l'aspetto significativo delle struct è determinato dalla possibilità di memorizzare informazioni di natura diversa al'interno di un'unica variabile. Una struct può essere utilizzata per integrare un gruppo di variabili che formano un'unità coerente di informazione.

Ad esempio il Linguaggio C non possiede un tipo fondamentale per rappresentare i numeri complessi: una soluzione semplice consiste nell'utilizzare una struct e nel definire un insieme di funzioni per la manipolazione di variabili. Si consideri a tale scopo il seguente esempio:

```
struct s_complesso
 {
  float reale;
  float immaginaria;
 };
```

A questo punto è possibile definire due variabili:

```
struct s_complesso a, b, c;
```

È possibile effettuare operazioni di assegnamento tra i vari campi della struct come ad esempio:

```
a.reale = b.reale;
```

D'altra parte non si può scrivere un'espressione del tipo `c = a + b`, per la quale è necessario invece scrivere:

```
c.reale = a.reale + b.reale;
c.immaginaria = a.immaginaria + b.immaginaria;
```

A questo punto potrebbe quindi essere conveniente scriversi un insieme di funzioni che effettuino le operazioni elementari sui numeri complessi da richiamare ogni volta.

Strutture e funzioni

La maggior parte dei compilatori C consente di passare a funzioni e farsi restituire come parametri intere strutture. Se si desidera che una funzione possa cambiare il valore di un parametro è necessario passarne il puntatore.

```
struct s_complesso somma(struct s_complesso a , struct s_complesso b)
{
 struct s_complesso c;
 c.reale = a.reale + b.reale;
 c.immaginaria = a.immaginaria + b.immaginaria;
 return (c);
}
```

Definita la funzione somma è possibile chiamarla, come nel seguente esempio:

```
struct s_complesso x, y, z;
...
x = somma(y, z);
```

Si tenga presente che il passaggio di una struct per valore può richiedere un elevato quantitativo di memoria.

Puntatori a strutture

Come per tutti i tipi fondamentali è possibile definire un puntatore ad una struct.

```
struct s_dipendente * ptr
```

definisce un puntatore ad una struttura s_dipendente. Il funzionamento è pressoch è inalterato:

```
(*ptr).anni
```

è il campo anni della struttura s_dipendente a cui punta ptr, ed è un numero intero. È necessario utilizzare le parentesi in quanto il punto `.` ha una priorità superiore all'asterisco `*`. Di fatto l'utilizzo di puntatori a struct è estremamente comune e la combinazione della notazione `*` e `.` è particolarmente prona ad errori; esiste quindi una forma alternativa più diffusa che equivale a `(*ptr).anni`, ed è la seguente:

```
prt->anni
```

Questa notazione dà un'idea più chiara di ciò che succede: `prt` punta (i.e. `->`) alla struttura e `.anni` "preleva" il campo di interesse.

L'utilizzo di puntatori consente di riscrivere la funzione di somma di numeri complessi passando come parametri non le struct quanto i puntatori a queste.

```
void s_complesso somma(struct s_complesso *a , struct s_complesso *b , st
{
  c->reale = a->reale + b->reale;
  c->immaginaria = a->immaginaria + b->immaginaria;
}
```

In questo caso c è un puntatore e la chiamata deve essere fatta così:

```
somma(&x, &y, &z);
```

In questo caso si risparmia spazio nella chiamata alla funzione, in quanto si passano i tre indirizzi invece delle strutture intere

Array di strutture

Uno dei punti id forza del C è la capacità di combinare insieme tipi fondamentali e tipi derivati per ottenere strutture dati complesse a piacere, in grado di modellare entità dati del mondo reale. Si consideri il seguente esempio:

```
struct s_automobile
{
  char marca[50];
  char modello[70];
  int venduto;
};
typedef struct s_automobile auto;
```

Per poter gestire le informazioni di un concessionario è a questo punto necessario poter dichiarare delle variabili che memorizzino i dati relativi alle automobili vendute. Si ipotizzi che ci siano al più cento diverse combinazioni di marche e modelli da dover gestire. Nell'ambito del programma sarà quindi necessario disporre di 100 elementi di tipo auto e a tal scopo verrà dichiarato un array, come segue:

```
void main()
{
  auto concessionario[100];
  int i;
  ...
}
```

In questo modo si dichiara un array di `struct s_automobile` di cento elementi. Ogni elemento dell'array ha i suoi campi `marca`, `modello` e `venduto`, a cui si accede come segue:

```
...
printf("inserisci il nome della marca: \n");
gets(concessionario[i].marca);
concessionario[i].venduto=0;
...
```

I campi delle struct dei singoli elementi dell'array vengono poi trattati normalmente, in base al loro tipo (nell'esempio rispettivamente come un array di caratteri ed un intero)

Definizione di un nuovo tipo per le strutture

La dichiarazione della struct per poter gestire insiemi di dati non omogenei viene spesso completata introducendo un nuovo tipo, definito appunto dall'utente, che si va ad affiancare ai tipi fondamentali del C. Si consideri il caso della struct per la gestione dei numeri complessi. Al fine di evitare di dover scrivere ogni volta che si dichiara una variabile struct s_complesso è possibile definire un nuovo tipo apposito:

typedef struct s_complesso complesso;

In questo modo abbiamo introdotto un nuovo tipo che si affianca ad `int`, `char`, ... che si chiama complesso ed è possibile utilizzarlo nella dichiarazione di variabili, come mostrato di seguito:

```
struct s_complesso
{
  float reale;
```

```
  float immaginaria;
};
typedef struct s_complesso complesso;
...
void main()
{
   ...
   int a, b;
   complesso x, y;
}
```

Frequentemente la dichiarazione del tipo mediante l'istruzione `typedef` viene fatta concorrentemente alla dichiarazione della struct, secondo la seguente sintassi:

```
typedef struct nome_struttura
{
  lista dei campi (tipo-nome)
} nome_tipo_struttura;
```

La dichiarazione è più compatta. Riportata all'esempio precedente, il codice che si ottiene è il seguente:

```
typedef struct s_complesso
{
  float reale;
  float immaginaria;
} complesso;
```

Strutture e liste concatenate

Le strutture vengono utilizzate per la realizzazione del tipo elementare che costituisce l'elemento di base delle liste dinamiche concatenate, discusse nella Lezione 15.

T Teaching (http://home.deib.polimi.it/bolchini/classes.htm)
Fondamenti di Informatica (http://home.deib.polimi.it/bolchini/didattica/fi/)
Reti Logiche (http://home.deib.polimi.it/bolchini/didattica/rl/)
Dependable Systems (http://home.deib.polimi.it/bolchini/didattica/ds/)

R Research
Dependable Systems Design Methodologies (http://home.deib.polimi.it/bolchini/research/dsddm.html)
Context-Awareness & Personalization (http://home.deib.polimi.it/bolchini/research/pedigree.html)
S A V E (http://home.deib.polimi.it/bolchini/research/save.html)

P Publications (http://home.deib.polimi.it/bolchini/publications.htm)
Journals (http://home.deib.polimi.it/bolchini/articles.html)
Proceedings (http://home.deib.polimi.it/bolchini/proceedings.html)
Book chapters (http://home.deib.polimi.it/bolchini/inbooks.html)
Editorial (http://home.deib.polimi.it/bolchini/editorial.html)

O Other infos
colophon (http://home.deib.polimi.it/bolchini/colophon.html)