Azzolini Riccardo 2021-03-03

Puntatori

1 Puntatori

Un **puntatore** è una variabile il cui contenuto viene interpretato come un indirizzo in memoria. I puntatori permettono l'accesso diretto alla memoria, la visione diretta della macchina sottostante, dunque consentendo l'uso del linguaggio C a basso livello, come se fosse una sorta di "assembly strutturato".

Dato un qualunque tipo `T`, per definire una variabile `pt` come puntatore a un oggetto di tipo `T` si usa la sintassi:[1]

```
T *pt;
```

Specificare il tipo `T` degli oggetti a cui `pt` punta è necessario non tanto per determinare la dimensione della variabile `pt` stessa (solitamente, i puntatori a oggetti di tutti i tipi hanno una dimensione uguale, corrispondente alla dimensione del bus degli indirizzi della macchina), ma piuttosto per sapere come interpretare il contenuto presente in memoria all'indirizzo puntato.

2 Left value e right value

Quando una variabile compare in un assegnamento, assume un ruolo diverso a seconda che si trovi a sinistra o a destra dell'operatore `=`. Infatti, un assegnamento (ad esempio) `a = b` fa uso:

- dell'*indirizzo* della variabile `a`, che determina dove debba essere salvato il valore assegnato;
- del *contenuto* della variabile `b`, che è il valore da assegnare.

Allora, si dice che ogni variabile ha due "valori", che prendono il nome dai lati dell'assegnamento per cui sono usati:

- il **left value**, che è il suo indirizzo;
- il **right value**, che è il suo contenuto.

[1] Gli spazi prima/dopo l'asterisco non importano, quindi le definizioni dei puntatori si trovano comunemente scritte nelle forme `T *pt`, `T* pt` e `T * pt`, che sono del tutto equivalenti.

Solo alcuni oggetti, tra cui appunto le variabili, sono dotati di un left value. Un oggetto che non ha un left value, come ad esempio una costante numerica o un'espressione aritmetica, non può comparire nel lato sinistro di un assegnamento.

3 Operatore di estrazione di indirizzo

Per poter assegnare a un puntatore l'indirizzo di un oggetto in memoria, serve un modo di conoscere tale indirizzo. A tale scopo, il C fornisce l'**operatore di estrazione di indirizzo**, un operatore unario prefisso indicato con il simbolo `&`, che restituisce l'indirizzo dell'oggetto a cui viene applicato. Tale indirizzo può poi essere assegnato a una variabile di tipo puntatore. Ad esempio, al termine del frammento di codice

```
int i;
int *pt;
pt = &i;
```

la variabile `pt` contiene l'indirizzo della variabile `i`.

L'operatore `&` può essere applicato solo a oggetti dotati di indirizzo (left value). Ad esempio:

- se `a` è una variabile, allora `&a` è valido (e restituisce, come già detto, l'indirizzo della variabile `a`);
- se `A` è un array, è valido scrivere `&A[i]` per ottenere l'indirizzo dell'$(\texttt{i} + 1)$-esimo elemento di `A`;
- `&12` non è valido (dà un errore in compilazione), perché una costante numerica come `12` non ha un indirizzo associato;
- analogamente, `&(i + 12)` non è valido, perché l'espressione `i + 12` non ha un indirizzo.

4 Operatore di indirizzamento indiretto

Una volta assegnato un indirizzo a un puntatore, si può accedere all'oggetto situato a tale indirizzo tramite l'**operatore di indirizzamento indiretto** (**dereferencing**), che è un operatore unario prefisso indicato con il simbolo `*` (non a caso, lo stesso simbolo usato nella definizione di un puntatore). Ad esempio, nel frammento di codice

```
int i;
int *pt;
pt = &i;
*pt = 16;
```

il puntatore `pt` contiene l'indirizzo della variabile `i`, quindi l'assegnamento `*pt = 16` equivale a `i = 16`.

Si parla di "indirizzamento indiretto" perché un uso dell'operatore `*` comporta due accessi alla memoria: prima si accede alla variabile puntatore per leggere l'indirizzo che essa contiene, dopodiché si accede a tale indirizzo per leggere o modificare l'oggetto corrispondente.

L'operatore `*` può essere considerato come l'inverso dell'operatore `&`:

- se `a` è una variabile qualsiasi, `*&a` equivale ad `a`;
- se `pt` è un puntatore, `&*pt` equivale a `pt`.

5 Tipi degli oggetti puntati

Come detto prima, quando si definisce un puntatore si specifica il tipo di oggetto a cui esso punta. Allora, è lecito assegnare al puntatore solo gli indirizzi di oggetti di tale tipo. Ad esempio, date le variabili

```
int *pt;
int a;
float b;
```

- l'assegnamento `pt = &a` è valido, perché `&a` è l'indirizzo di un intero;
- `pt = &b` non è valido, perché `&b` è l'indirizzo di un float, mentre con `*pt` si interpreterebbero i byte del valore di `b` come se rappresentassero un numero intero.

Tipicamente, l'assegnamento di un indirizzo di tipo errato a un puntatore non genera un errore in compilazione, ma solo un'avvertenza (a seconda del livello di warning al quale si è impostato il compilatore). Inoltre, tale avvertenza può essere eliminata inserendo una conversione di tipo esplicita (*cast*), come ad esempio:

```
int *pt;
float b;
pt = (int*)&b;
```

Si noti che l'aggiunta di questa conversione non altera in alcun modo il comportamento del programma: semplicemente, essa indica al compilatore che il programmatore è consapevole del fatto che i tipi non coincidano, e vuole "assumersi la responsabilità" di eventuali malfunzionamenti.