Azzolini Riccardo 2021-03-08

Puntatori

1 Aritmetica dei puntatori

Gli operatori binari di somma e sottrazione possono essere applicati, con un'interpretazione particolare, quando uno degli operandi è un puntatore e l'altro è un intero. Dati un puntatore *T* `*pt` e un'espressione *int_expr* che assume valori interi, il valore dell'espressione `pt + int_expr` è l'indirizzo ottenuto sommando all'indirizzo contenuto in `pt` il numero di byte richiesti da *int_expr* variabili di tipo *T*, ovvero il numero di byte richiesti da una variabile di tipo *T* moltiplicato per il valore dell'espressione *int_expr*. Un analogo calcolo avviene per la sottrazione.

Ad esempio, supponendo che una variabile `char` richieda 1 byte e che una variabile `long` richieda 8 byte, al termine di questo frammento di codice:

```
char *pt1;
long *pt2;
char a;
long b;
pt1 = &a + 2;
pt2 = &b + 2;
```

- `pt1` punta al secondo byte dopo `a`;
- `pt2` punta al sedicesimo byte dopo il primo byte di `b`.

1.1 Operatore `sizeof`

L'operatore `sizeof` permette di conoscere il numero di byte necessari per rappresentare il valore di un'arbitraria espressione o di un determinato tipo. Esso ha la sintassi

`sizeof(`*expr*`)`

dove *expr* può essere un'espressione o un tipo.

Ad esempio, se una variabile di tipo `int` richiede 4 byte, allora il frammento di codice

```
int x;
printf("%lu\n", sizeof(int));
printf("%lu\n", sizeof(x));
printf("%lu\n", sizeof(x + 1));
```

stampa tre volte il valore 4, perché l'operatore `sizeof` viene usato specificando prima il tipo `int`, e poi delle espressioni (`x` e `x + 1`) che assumono valori di tipo `int`.

Grazie all'operatore `sizeof`, si può descrivere in modo più sintetico l'indirizzo calcolato espressione *`pt + int_expr`*: esso è l'indirizzo ottenuto sommando *`int_expr`*·`sizeof(`*`T`*`)` byte all'indirizzo contenuto in *`pt`*.

2 Corrispondenza tra puntatori e vettori

A basso livello, un array è rappresentato da una sequenza di locazioni di memoria adiacenti. Allora, se si assegna a un puntatore l'indirizzo di un elemento di un array, è possibile accedere all'array tramite tale puntatore. Ad esempio, nel codice

```
unsigned int voti[20];
unsigned int *pt;
pt = &voti[9];
*pt = 18;
```

si assegna a `pt` l'indirizzo del decimo elemento (quello con indice 9) di `voti`, quindi la successiva istruzione `*pt = 18;` equivale a `voti[9] = 18;`.

Il linguaggio C consente di interpretare il nome di un array come un puntatore *non modificabile* contenente l'*indirizzo di base* dell'array, cioè l'indirizzo del primo elemento: l'assegnamento `pt = voti;` equivale a `pt = &voti[0];`. Inoltre, l'operatore `[]` di accesso a un array può essere usato anche con i puntatori, e gli operatori `*` e `+` possono essere usati anche con gli array. Ad esempio,

- `pt[5]` equivale a `*(pt + 5)`,
- `*(voti + 9)` equivale a `voti[9]`,

quindi il seguente frammento di codice modifica correttamente gli elementi di indice 5 e 9 del vettore `voti`:

```
pt = voti;
pt[5] = 30;
*(voti + 9) = 20;
```

La notazione *`pt`*`[`*`i`*`]` è forse la più leggibile delle due, quindi è quella più usato in pratica (sia per gli array che per i puntatori), ma la notazione `*(`*`pt + i`*`)` è interessante perché mette in luce esattamente la semantica operazionale dell'accesso a un array. Ad esempio, data la variabile `pt` dei frammenti di codice precedenti, per l'istruzione `*(pt + 9) = 30;` (o `pt[9] = 30;`) il compilatore genera del codice che:

1. preleva l'indirizzo contenuto in `pt`;
2. somma $9 \cdot$ `sizeof(unsigned int)` byte a tale indirizzo;

3. interpreta le celle di memoria a partire dall'indirizzo risultante come una variabile di tipo `unsigned int`, alla quale assegna il valore 30.

La corrispondenza tra puntatori e array è sostanzialmente il motivo per cui l'indice del primo elemento di un array è 0. Infatti, se si partisse da 1, l'accesso `A[i]` dovrebbe corrispondere a `*(A + i - 1)`, in modo che `A[1]` corrisponda a `*(A + 1 - 1)`, cioè `*A`, dato che l'indirizzo di `A` è già quello del primo elemento. Invece, facendo iniziare gli indici da 0 si può risparmiare una sottrazione ogni volta che si accede a un array.