Array e puntatori

Array

Quando si usa una espressione di tipo array di `T`, viene applicato il *type decay* e si ottiene il puntatore al primo elemento dell'array. Questa conversione (trasformazione di lvalue) è necessaria per evitare l'uso di copie costose quando un array viene passato come argomento ad una funzione: si passa (per valore) il puntatore.

Il legame tra array e puntatori è molto forte: basti considerare che la sintassi dell'indicizzazione di un array non è altro che una abbreviazione per un utilizzo semplificato dell'aritmetica dei puntatori.

Esempio

Si considerino le due variabili:

```
int a[100];
int b = 5;
```

L'espressione `a[b]` è in tutto e per tutto equivalente all'espressione `*(a + b)`. In questa seconda espressione si usa infatti l'array `a`, che decade al puntatore al suo primo elemento; poi si somma `b` al puntatore, che corrisponde a muoversi in avanti nell'array di 5 posizioni, ottenendo l'indirizzo del sesto elemento. Infine si dereferenzia l'indirizzo, ottenendo il sesto elemento dell'array.

Siccome la somma è commutativa, lo stesso risultato lo si ottiene anche usando l'espressione `*(b + a)` che per quanto detto sopra risulta essere equivalente a `b[a]`.

> NOTA BENE: chiaramente, un programmatore sensato dovrebbe astenersi dall'utilizzare costrutti che hanno il solo scopo di sorprendere (o trarre in inganno).
>
> NOTA: questo "trucco" funziona solo con gli array; non funziona con altri contenitori, quali `std::vector<T>`.

Torna all'indice

Aritmetica dei puntatori

Se `ptr` è un puntatore (del tipo corretto) che indirizza un elemento all'interno di un array, allora le espressioni

```
ptr + n   // muoviti in avanti di n posizioni
n + ptr   // idem
ptr - n   // muoviti all'indietro di n posizioni
```

sono legittime nella misura in cui il puntatore risultante dopo il movimento continui a puntare ad un elemento dell'array (cioè non si è andati oltre il limite iniziale o finale) oppure punti all'indirizzo immediatamente successivo alla fine dell'array.

Dati due puntatori `ptr1` e `ptr2` (del tipo corretto) che indirizzano elementi dello stesso array, l'espressione `ptr1 - ptr2` indica la distanza tra i due puntatori, ovvero il numero di elementi che li separa (si noti che la distanza potrebbe essere negativa).

L'aritmetica dei puntatori si presta alla definizione di un importante idioma di programmazione relativo all'iterazione su array.

Torna all'indice

Esempio

```
int a[100];

// iterazione basata su indice
for (int i = 0; i != 100; ++i) {
 // fai qualcosa con a[i]
}

// iterazione basata su puntatore
for (int* p = a; p != a + 100; ++p) {
 // fai qualcosa con *p
 // I puntatori sono iteratori per gli array
}
```

La seconda forma si presta bene a generalizzazioni che non richiedono di conoscere il punto di inizio dell'array e la sua dimensione. Se sono sicuro che `p1` e `p2` sono puntatori validi sull'interno dell'array e sono anche sicuro che `p1` non viene dopo `p2`, allora posso iterare su tutti gli elementi compresi tra l'elemento puntato da `p1` (incluso) e l'elemento puntato da `p2` (escluso), nel modo seguente:

```
// iterazione basata su coppie di puntatori
for ( ; p1 != p2; ++p1) {
 // fai qualcosa con *p1
}
```

Spesso, per la coppia `p1` e `p2` si usano i nomi `first` e `last`, con l'accortezza di ricordarsi che `last`, in effetti, si riferisce alla posizione *successiva* all'ultimo elemento che si vuole processare.

Se si vuole specificare una sequenza vuota, è sufficiente fornire una coppia di puntatori identici (ottenendo quindi un ciclo che non effettua alcuna iterazione). Questo i dioma è stato esteso nel C++ al caso degli iteratori sulle sequenze generiche e sui contenitori della libreria standard (e quindi è di estrema rilevanza per la programmazione in C++).

Torna all'indice