Tipi riferimento e tipi puntatore

I tipi riferimento e i tipi puntatore vengono spesso confusi tra loro dai programmatori che prendono in considerazione gli aspetti implementativi del linguaggio, in quanto entrambi sono rappresentati internamente utilizzando l'indirizzo dell'oggetto riferito o puntato.

Una visione a più alto livello mette in evidenza alcune differenze, sia a livello sintattico che (cosa più importante) a livello semantico. Un modo intuitivamente semplice (anche se formalmente non esatto) di capirne la differenza è il seguente:

- un PUNTATORE è un oggetto, il cui valore (un indirizzo) si può riferire ad un altro oggetto (necessariamente diverso);
- un RIFERIMENTO non è un oggetto vero e proprio, ma è una sorta di "nome alternativo" fornito per accedere ad un oggetto esistente.

  Un puntatore è una variabile che contiene l'indirizzo di un'altra variabile. Un riferimento è un'alias di un'altra variabile, ovvero un altro nome per la stessa area di memoria.

In sostanza, la principale differenza tra un puntatore e un riferimento è che un puntatore può essere inizializzato a `nullptr` e può essere ri-assegnato a puntare ad altre aree di memoria, mentre un riferimento deve essere inizializzato al momento della definizione e non può essere ri-assegnato ad un'altra area di memoria. Inoltre, la sintassi per accedere al valore contenuto in un'area di memoria è diversa: per un puntatore si usa l'operatore di dereferenziazione, mentre per un riferimento si usa il riferimento stesso come se fosse la variabile originale.

Ritorna all'indice

Osservazioni

Da queste "*definizioni*" seguono alcune osservazioni:

1. Quando viene creato un riferimento, *questo deve sempre essere inizializzato*, in quanto si deve sempre riferire ad un oggetto esistente; in altre parole, non esiste il concetto di "riferimento nullo". In contrasto, un puntatore può essere inizializzato con il letterale `nullptr` (o con il puntatore nullo del tipo corretto, mediante conversioni implicite), nel qual caso NON punterà ad alcun oggetto.

2. Una volta creato un riferimento, questo si riferirà sempre allo stesso oggetto; non c'è modo di "riassegnare" un riferimento ad un oggetto differente. In contrasto, durante il suo tempo di vita, un oggetto puntatore (che non sia qualificato `const`) può essere modificato per puntare ad oggetti diversi o a nessun oggetto.

3. Ogni volta che si effettua una operazione su un riferimento, in realtà si sta (implicitamente) operando sull'oggetto riferito. In contrasto, nel caso dei puntatori abbiamo a che fare con due oggetti diversi: l'oggetto puntatore e l'oggetto puntato. Eventuali operazioni di lettura e scrittura (comprese le operazioni relative all'aritmetica dei puntatori) applicate direttamente al puntatore accederanno e potenzialmente modificheranno l'oggetto puntatore. Per lavorare sull'oggetto puntato, invece, occorrerà usare l'operatore di dereferenziazione, in una delle forme

```
*p     // operator* prefisso
p->a   // operator-> infisso, equivalente a (*p).a
```

4. Un eventuale qualificatore 'const' aggiunto al riferimento si applica necessariamente all'oggetto riferito e non al riferimento stesso. In contrasto, nel caso del puntatore, avendo due oggetti (puntatore e puntato), è possibile specificare il qualificatore `const` (o meno) per ognuno dei due oggetti. A livello sintattico, nella dichiarazione di un puntatore è possibile scrivere due volte il qualificatore const:
   - se `const` sta a sinistra di `*`, si applica all'oggetto puntato;
   - se `const` sta a destra di `*`, si applica all'oggetto puntatore.

Ritorna all'indice

Esempi

```
int i = 5;          // oggetto modificabile
const int ci = 5;   // oggetto non modificabile

int& r_i = i;           // ok: posso modificare i usando r_i
const int& cr_i = i;    // ok: prometto di non modificare i usando cr_i
int& r_ci = ci;         // errore: non posso usare un riferimento non-const
                        // per accedere ad un oggetto const

const int& cr_ci = ci; // ok: accesso in sola lettura

int& const cr = i;     // errore: const può stare solo a sinistra di &

int* p_i;              // p_i e *p_i sono entrambi modificabili
const int* p_ci;       // p_ci è modificabile, *p_ci non lo è
int* const cp_i = &i;         // cp_i non è modificabile, *cp_i è modificabile
const int* const cp_ci = &i; // cp_ci e *cp_ci sono entrambi non modificabili

p_i = &i;      // ok
p_i = &ci;     // errore: non posso inizializzare un puntatore a non-const
               // usando un indirizzo di un oggetto const

p_ci = &i;     // ok: prometto che non modificherò i usando *p_ci
p_ci = &ci;    // ok

cp_i = nullptr;   // errore: cp_i è non modificabile
cp_ci = &ci;      // errore: cp_ci è non modificabile
```

Ritorna all'indice

Somiglianze

Evidenziate le differenze, possiamo ora discutere alcune somiglianze (magari non tanto banali) tra puntatori e riferimenti.

1. Quando termina il tempo di vita di un puntatore (ad esempio, quando si esce dal blocco di codice nel quale era stato definito come variabile locale), viene distrutto l'oggetto puntatore, ma NON viene distrutto l'oggetto puntato (cosa che potrebbe dare origine ad un memory leak). Analogamente, quando un riferimento va fuori scope, l'oggetto riferito non viene distrutto.

   NOTA - Il caso speciale Esiste però il caso speciale del riferimento inizializzato con un oggetto temporaneo, che viene distrutto insieme al riferimento stesso (se ne era parlato nella discussione sul tempo di vita degli oggetti).

2. Analogamente al *dangling pointer* (un puntatore non nullo che contiene l'indirizzo di un oggetto non più esistente), è possibile creare un dangling reference, ovvero un riferimento che si riferisce ad un oggetto non più esistente. Il classico esempio è il seguente:

```
struct S { /* ... */ };

S& foo() {
  S s;
  // ...
  return s;
}
```

Si tratta chiaramente di un ***GRAVE ERRORE*** di programmazione: la funzione ritorna per riferimento un oggetto che è stato allocato automaticamente all'interno della funzione stessa; tale oggetto però, viene automaticamente distrutto quando si esce dal blocco nel quale è stato definito e quindi il riferimento restituito al chiamante è invalido. L'approccio corretto è di modificare l'interfaccia della funzione `foo` affinché ritorni per valore, invece che per riferimento.

Ritorna all'indice