

# 06\_References\_vs\_Pointers

Table of contents

- [Riferimenti vs Puntatori](#)
- [Riferimenti & Puntatori](#)

## Riferimenti vs Puntatori

<code>int&amp; ref</code>	<code>int* pref = &amp;ref</code>
Un reference	Un puntatore

Le differenze più importanti sono le seguenti.

1. Quando viene creato un riferimento, dobbiamo *per forza inizializzarlo*; per il puntatore possiamo anche non farlo (wild/dangling/zero).
2. Creato il riferimento, per tutta la sua vita si *referirà a un specifico oggetto*, quindi non posso creare un riferimento e cambiare l'oggetto a cui si riferisce; i puntatori possono puntare oggetti diversi.
3. Con operazioni su riferimento, lavoriamo sempre sull'oggetto riferito; il puntatore è un oggetto diverso e abbiamo 2 tipologie di operazioni possibili: *operazioni sul puntatore*, *operazioni sull'oggetto puntato*.

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

4. Quando usiamo `const` a chi ci stiamo riferendo?  
Tutto quello che sta a sinistra si riferisce all'oggetto puntato, tutto quello che sta sulla destra si riferisce al puntatore.

```
int i = 5;  
const int ci = 5; // CORRECT: non modificabile  
int& r_i = i; // CORRECT: posso modificare 'i' con 'r_i'  
const int& cr_i = 1; // CORRECT: posso modificare 'i' usando 'cr_i'  
  
int& r_ci = ci; //ERROR: riferimento non 'const' ad oggetto 'const'  
  
const int& cr_ci = ci; // CORRECT: accesso in sola lettura
```

```
int* p_i; // CORRECT: entrambi 'p_i' e '*p_i' sono modificabili  
const int* p_ci; // CORRECT: puntatore costante  
int* const cp_i = &i; // CORRECT: 'cp_i' non modificabile, '*cp_i' modificabile  
const int* const cp_ci = &i; // niente è modificabile
```

## Riferimenti & Puntatori

1. Al termine del tempo di vita di un puntatore, cosa accade?  
Il puntatore quando muore *non fa succedere nulla all'oggetto puntato*, cosa a cui dobbiamo porre attenzione per memory leak.  
Al termine del tempo di vita di un riferimento, anche lui *non tocca l'oggetto a cui si riferisce*.

2. Non esistono i riferimenti nulli, ma possono esistere i *riferimenti dangling*: il riferimento sopravvive all'oggetto riferito che invece scompare.

```
// restituiamo il riferimento di un
// oggetto che oramai è stato distrutto
struct S { /* ... */ };
S& foo {
    S s;
    // ...
    return s;
}
```

L'esempio sopra è un classico esempio di copia temporanea non necessaria: viene creato un oggetto di tipo `S` che alla fine non serve niente se non a essere distrutto. Da questa idea nascono nel C++, per ovviare al problema di copie inutili, *riferimenti a r-value* e *costruttori di spostamento*.

---

 metodologie >  CODE >  Pointer\_vs\_Reference >  PointerReference.cpp

---

01/03/2023