9 - Array e puntatori

Type decay

Quando si usa un'espressione che rappresenta un array viene applicato il **type decay**, ovvero l'array decade in un puntatore al suo primo elemento (tecnicamente, una trasformazione di lvalue). Questa conversione è utile per evitare copie costose degli array.
Gli array non possono essere copiati direttamente; passandoli a una funzione, viene passato per valore solo il puntatore al primo elemento.

```
void func (int* p) { /* ... */ }

int a[10];
func(a); // 'a' decade in un puntatore al primo elemento
```

L'indicizzazione degli array è, in realtà, un'abbreviazione che sfrutta l'aritmetica dei puntatori.

```
int a[100];
int b = 5;
int x = a[b]; //accedi all'elemento in posizione b dell'array
int y = *(a + b); //equivalente: somma b al puntatore 'a' e dereferenzia
```

- l'array 'a' decade in un puntatore al primo elemento (`&a[0]`);
- 'b' viene sommato al puntatore: ci si sposta in avanti di 'b' elementi nell'array;
- l'operatore `*` dereferenzia il puntatore per accedere al valore in memoria.
  Poiché la somma è commutativa, `*(a + b)` è equivalente a `*(b + a)` ; infatti `a[b]` è uguale a `b[a]` .
  Questo comportamento è valido solo per gli array 'classici'. Non funziona per altri tipi di contenitori, come `std::vector<T>` , perché non si basano su puntatori diretti.

Aritmetica dei puntatori

I puntatori supportano alcune operazioni aritmetiche. Se `ptr` è un puntatore che punta a un elemento di un array, possiamo:

1. **spostarsi in avanti** di n posizioni: `ptr + n` o `n + ptr` ;
2. **spostarsi indietro** di n posizioni: `ptr - n` .
   Queste operazioni funzionano a patto che il puntatore risultante:

- resti all'interno dell'array;
- oppure punti all'indirizzo immediatamente successivo alla fine dell'array (es. `a + 100` per un array di 100 elementi).

```
int a[100];
int* p1 = &a[10];  // Punta all'11° elemento
int* p2 = &a[5];   // Punta al 6° elemento
```

```
int distanza = p1 - p2; // distanza = 5
```

Iterazione con array e puntatori

Iterazione basata su indice

Il metodo più comune per iterare su un array è usare un indice:

```
int a[100];
for (int i = 0; i < 100; ++i) {
  // Fai qualcosa con a[i]
}
```

Iterazione basata su puntatore

Alternativa:

```
int a[100];
for (int* p = a; p != a + 100; ++p) {
  // Fai qualcosa con *p
}
```

Iterazione con coppie di puntatori

Se si conoscono due puntatori validi che delimitano un intervallo di un array (`p1` e `p2`), è possibile iterare senza conoscere esplicitamente l'inizio e la fine dell'array:

```
int* p1 = &a[10]; // Punta all'11° elemento
int* p2 = &a[20]; // Punta al 21° elemento

for (; p1 != p2; ++p1) {
  // Fai qualcosa con *p1
}
```

in questo caso si iterano solo gli elementi tra i due puntatori.
First e last
Quando si usa una coppia di puntatori per iterare, è comune chiamarli `first` (primo elemento incluso) e `last` (primo elemento escluso).
Se `first` e `last` coincidono, la sequenza è vuota e il ciclo non viene eseguito.

Iteratori generici

L'idioma basato su coppie di puntatori è stato generalizzato in c++ per funzionare con gli iteratori, che rappresentano una generalizzazione dei puntatori. Gli iteratori si usano con i contenitori della **Standard Template Library**, come `std::vector` , `std::list` , ecc.
Esempio con std::vector

```
#include <vector>

std::vector<int> v = {1, 2, 3, 4, 5};
```

```
//iterazione con iteratori
for (auto it = v.begin(); it != v.end(); ++it) {
    //fai qualcosa con *it
}
```

- `v.begin()` è un iteratore al primo elemento.
- `v.end()` è un iteratore alla posizione successiva all'ultimo elemento.

10 - Lvalue e Rvalue