12.1 – Algoritmi STL

Introduzione e algoritmi di ricerca

Capitoli:

- 21.1, 21.2

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Agenda

- Introduzione e panoramica sugli algoritmi STL
- Un primo esempio: `std::find()`

Algoritmi STL

- STL offre un grande numero di algoritmi (circa 80!)
- Impossibile conoscerli tutti, ma:
  - Conoscerne alcuni permette di abituarci alla sintassi
  - Buona base di partenza per usare anche altri algoritmi
- Conoscere gli algoritmi ci permette di risparmiare:
  - Tempo
  - Sforzi

https://en.cppreference.com/w/cpp/algorithm

Algoritmi STL

- Molti algoritmi disponibili:
  - Ricerca
  - Ordinamento
  - Conteggio
  - Copia con/senza duplicati
  - Fusione
- **Algoritmi compatibili con tutti i container STL**

Algoritmi STL – una selezione

Algoritmo	Significato
`r = std::find(b, e, v)`	`r` punta alla prima occorrenza di `v` in `[b:e)`
`r = std::find_if(b, e, p)`	`r` punta al primo elemento `x` in `[b:e)` tale che `p(x) == true`
`x = std::count(b, e, v)`	`x` è il numero di occorrenze di `v` in `[b:e)`
`x = std::count_if(b, e, p)`	`x` è il numero di elementi in `[b:e)` per cui `p(x) == true`
`std::sort(b, e)`	Ordina `[b:e)` usando `<`
`std::sort(b, e, p)`	Ordina `[b:e)` usando `p`
`std::copy(b, e, b2)`	Copia `[b:e)` in `[b2:b2+(e-b))` – è necessario sufficiente spazio a destinazione
`std::unique_copy(b, e, b2)`	Come sopra, elimina i duplicati
`std::merge(b, e, b2, e2, r)`	Fonde due sequenze ordinate `[b2:e2)` e `[b:e)` in `[r:r+(e-b)+(e2-b2))`
`r = std::equal_range(b, e, v)`	`r` è la sottosequenza del range ordinato `[b:e)` contenente `v` (ricerca binaria)

Algoritmi STL – numerici

Algoritmo	Significato
`std::equal(b, e, b2)`	Confronta tutti gli elementi in `[b:e)` e `[b2:b2+(e-b))`
`x = std::accumulate(b, e, i)`	`x = i +` la somma di tutti gli elementi in `[b:e)`
`x = std::accumulate(b, e, i, op)`	Come sopra, ma con la somma calcolata usando `op`
`x = std::inner_product(b, e, b2, i)`	Prodotto scalare di `[b:e)` e `[b2:b2+(e-i))`
`x = std::inner_product(b, e, b2, i, op, op2)`	Come sopra, ma con `op` e `op2` al posto di `+` e `*`

std::find()

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

std::find()

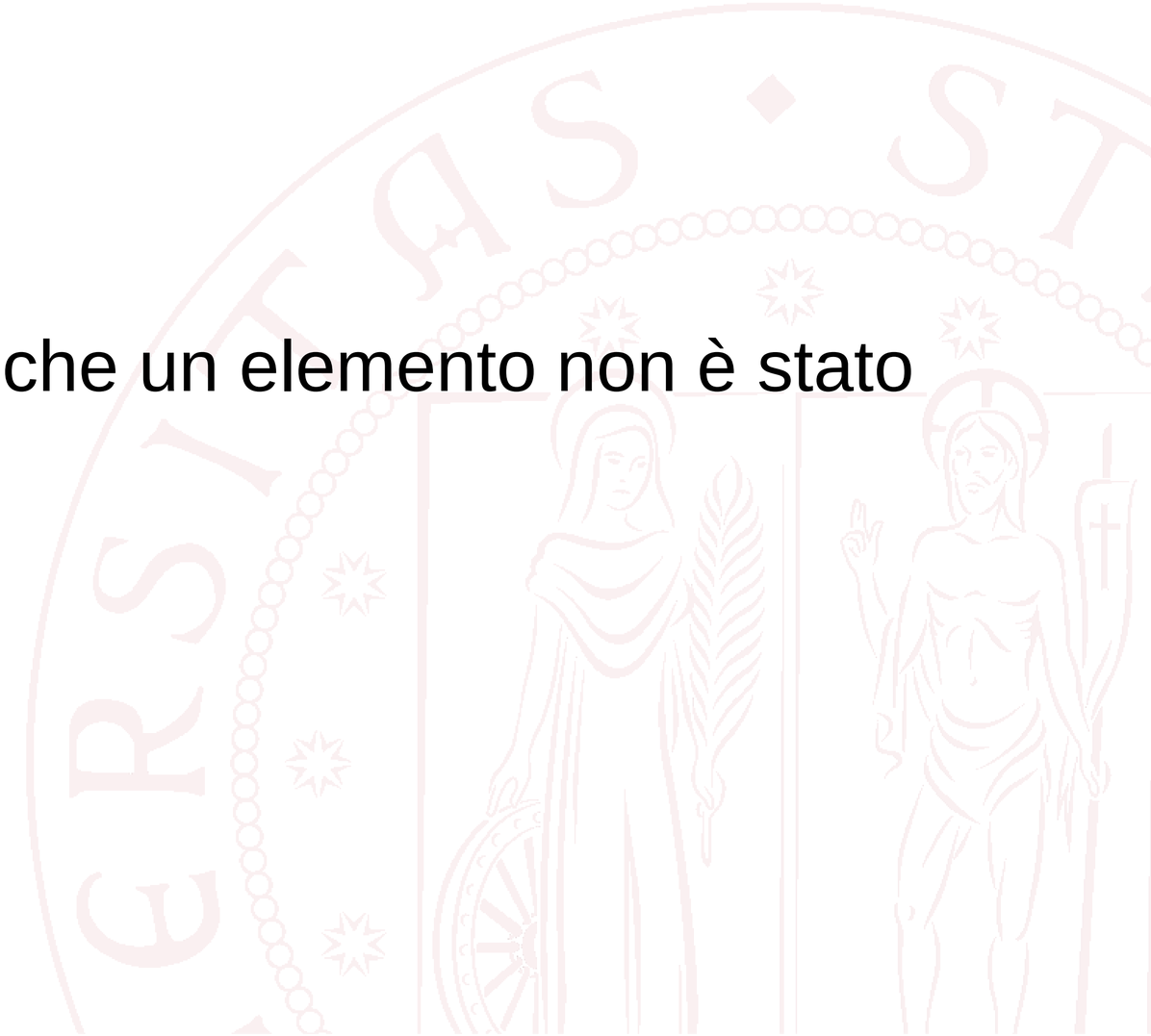
- `std::find()` cerca un elemento con un dato valore in una sequenza

```
template<typename In, typename T>
    // In: iteratore di input
    // T deve essere confrontabile
In find(In first, In last, const T& val)
{
    while(first != last && *first != val) ++first;
    return first;
}
```

- Implementazione efficiente e veloce

`std::find()`

- La sequenza su cui opera `std::find()` è definita mediante due iteratori
  - `[first, last)`
- Ritorna un iteratore
  - Alla prima occorrenza trovata
  - `end()` se il valore non è stato trovato
- In STL, ritornare **`end()`** spesso indica che un elemento non è stato trovato



Caratteristiche di `std::find()`

- `std::find()` è generico
  - Rispetto al tipo di container
  - Rispetto all'elemento contenuto
- La chiamata a `std::find()` non cambia in funzione di questi due elementi
- Flessibilità comune agli algoritmi STL
  - Effetto dell'**interfaccia standard** gestita tramite iteratori
    - Vedi esempio slide successiva

Flessibilità

```
void f(std::vector<int>& v, int x) {

    std::vector<int>::iterator p = std::find(v.begin(), v.end(), x);

    if (p != v.end()) {
        // x è stato trovato
    }
}
```

```
void f(std::list<std::string>& v, std::string x) {

    std::list<std::string>::iterator p = std::find(v.begin(), v.end(), x);

    if (p != v.end()) {
        // x è stato trovato
    }
}
```

Recap

- Introduzione agli algoritmi STL
- `std::find()`
- Caratteristiche di flessibilità di `std::find()`