10.1 – Standard Template Library (STL)

Intro, sequenze e iteratori

Libro di testo:

- Capitoli 20.1, 20.2, 20.3

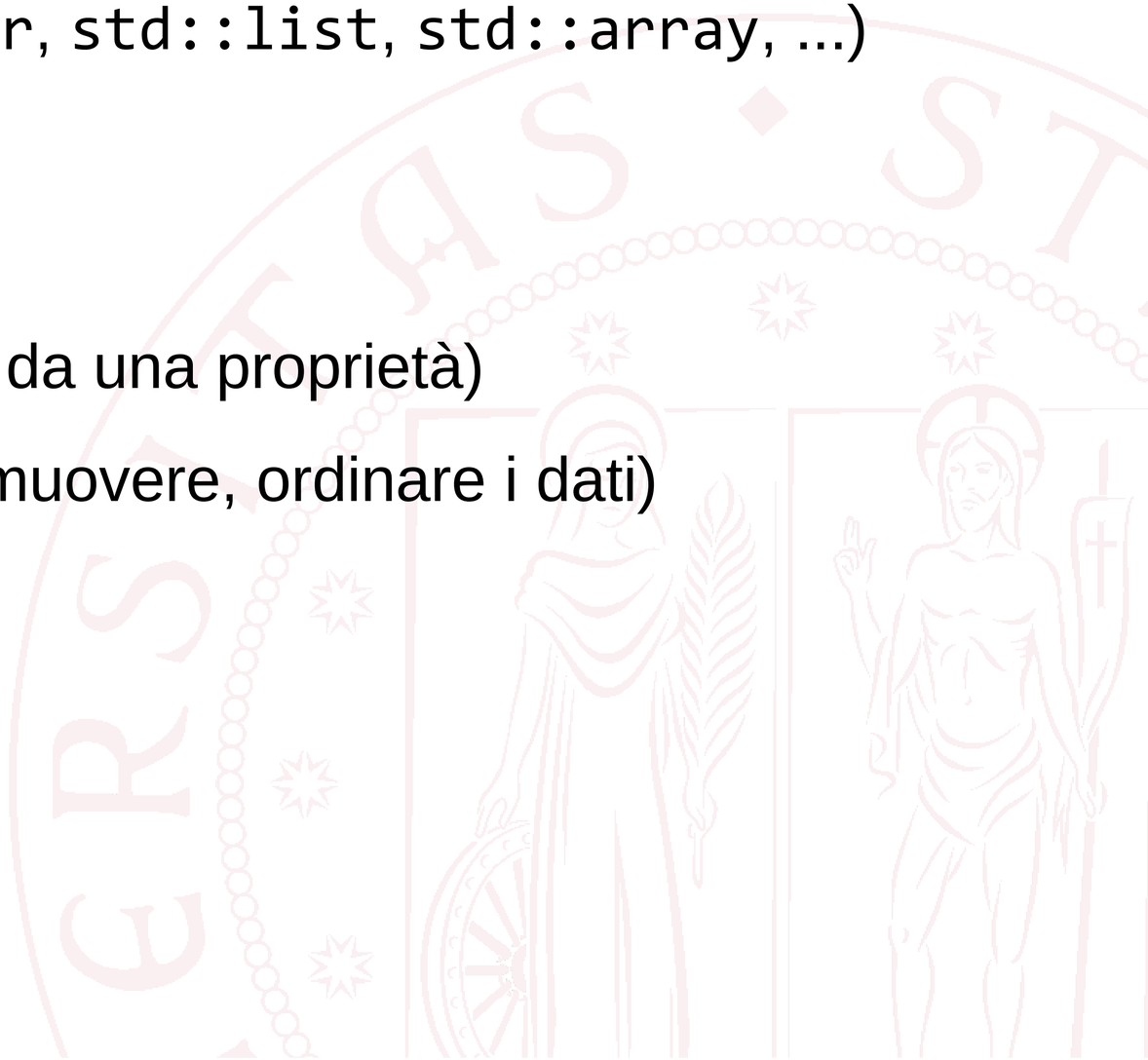
DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Agenda

- Gestire sequenze di dati
- STL: filosofia di progettazione
- Sequenze
- Iteratori
- Algoritmi e collezioni di dati

Operazioni sui dati

- Gestire una sequenze di dati implica numerose operazioni
  - Inserire i dati in **contenitori** (`std::vector`, `std::list`, `std::array`, ...)
  - Organizzare i dati
  - Stampa, accesso
  - Trovare i dati (da un indice, da un valore, da una proprietà)
  - Modificare un contenitore (aggiungere, rimuovere, ordinare i dati)



Operazioni sui dati

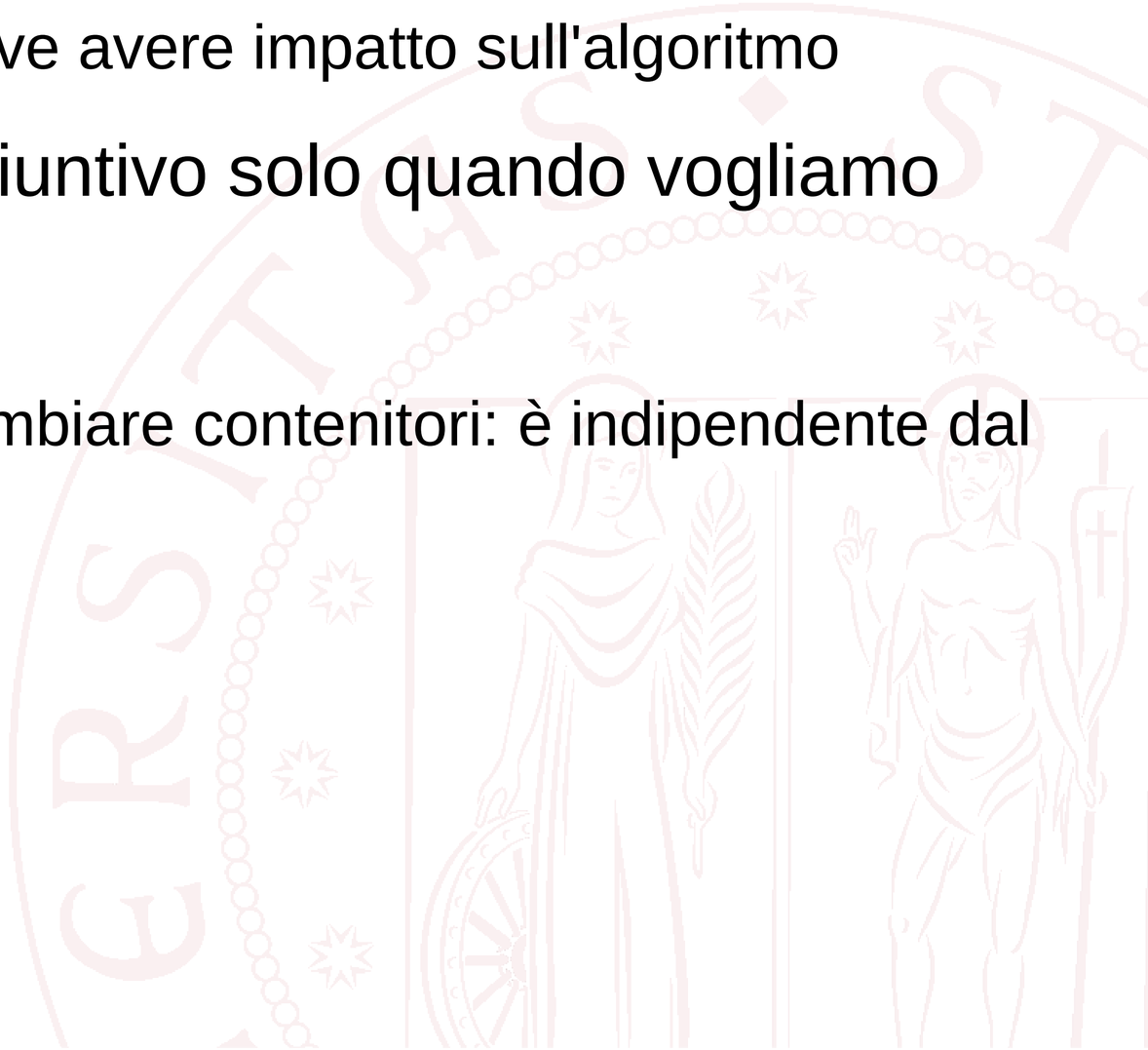
- Esempi:
  - Trovare la temperatura più alta
  - Trovare i valori più grandi di 8800
  - Trovare le differenze tra due sequenze
  - Calcolare la somma degli elementi
- Queste descrizioni prescindono dalla descrizione di come i dati sono memorizzati

STL: Standard Template Library

- STL: una parte della libreria standard che fornisce:
  - **Contenitori**
    - `std::vector`
    - `std::list`
    - `std::map`
    - …
  - **Algoritmi generici**
    - `std::sort`
    - `std::find`
    - `std::accumulate`

STL e dati

- I contenitori STL **sono molti**, ma hanno un **accesso unificato**
  - Modificare il tipo di un contenitore non deve avere impatto sull'algoritmo
- STL ci permette di scrivere codice aggiuntivo solo quando vogliamo implementare funzioni aggiuntive
  - Non quando vogliamo semplicemente cambiare contenitori: è indipendente dal tipo di dato contenuto
- Cosa serve per ottenere questo?



STL – desiderata

- Accesso uniforme ai dati
  - Indipendente da come sono salvati
  - Indipendente dal tipo usato
- Accesso safe
- Facilità di visita alle strutture dati
- Salvataggio compatto dei dati
- Aggiunta, rimozione e ricerca veloci
- Disponibilità degli algoritmi più comuni
  - copia, trova, cerca, ordina, somma…

Sequenze e iteratori

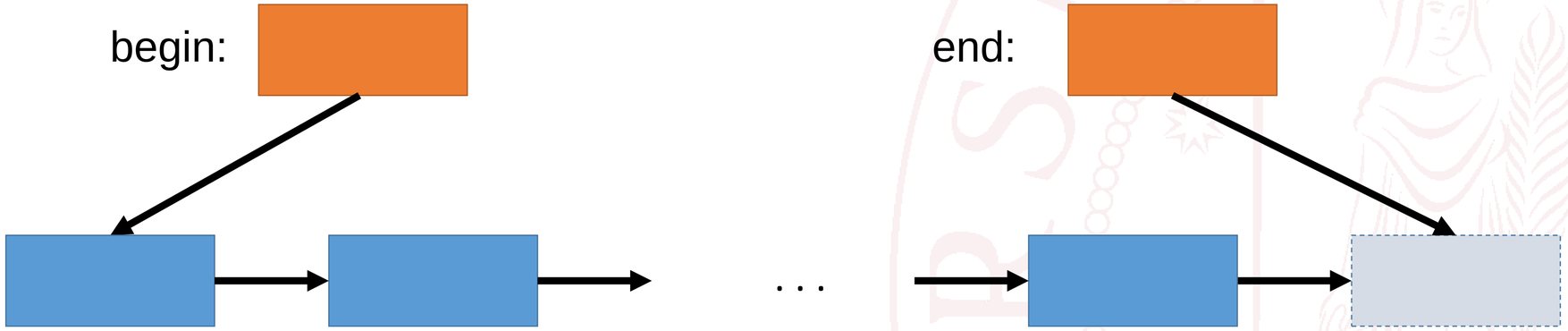
DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Sequenze e iteratori

- Il concetto fondamentale di STL è la ***sequenza***
- Una sequenza ha un inizio e una fine
- Una sequenza è gestita tramite **iteratori**
  - Un iteratore è un oggetto che identifica un elemento di una sequenza
  - Cosa ci ricorda?

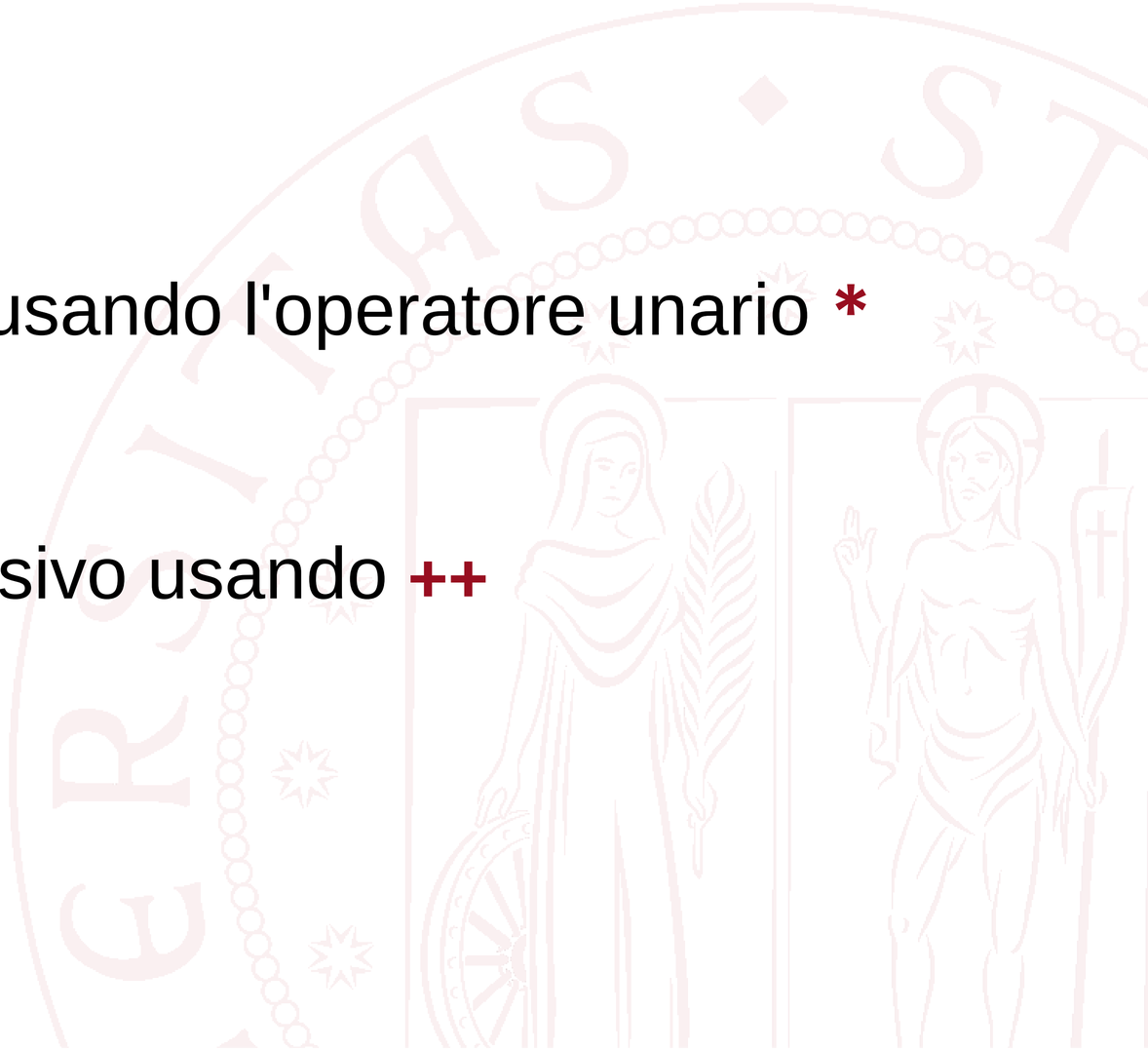
Iteratori

- Immaginiamo di avere una sequenza di elementi (aka contenitore)
- Iteratore `begin`: primo elemento
- Iteratore `end`: primo elemento dopo l'ultimo
- Sequenza: **`[begin, end)`**



Iteratori

- Un iteratore punta a un elemento della sequenza (o a uno dopo l'ultimo)
- È possibile confrontare due iteratori
  - Operatori: **==** e **!=**
- È possibile riferirsi agli oggetti puntati usando l'operatore unario ***** (dereference)
- È possibile ottenere l'elemento successivo usando **++**



Iteratori: operazioni standard

- p e q sono due iteratori della stessa sequenza (aka contenitore)

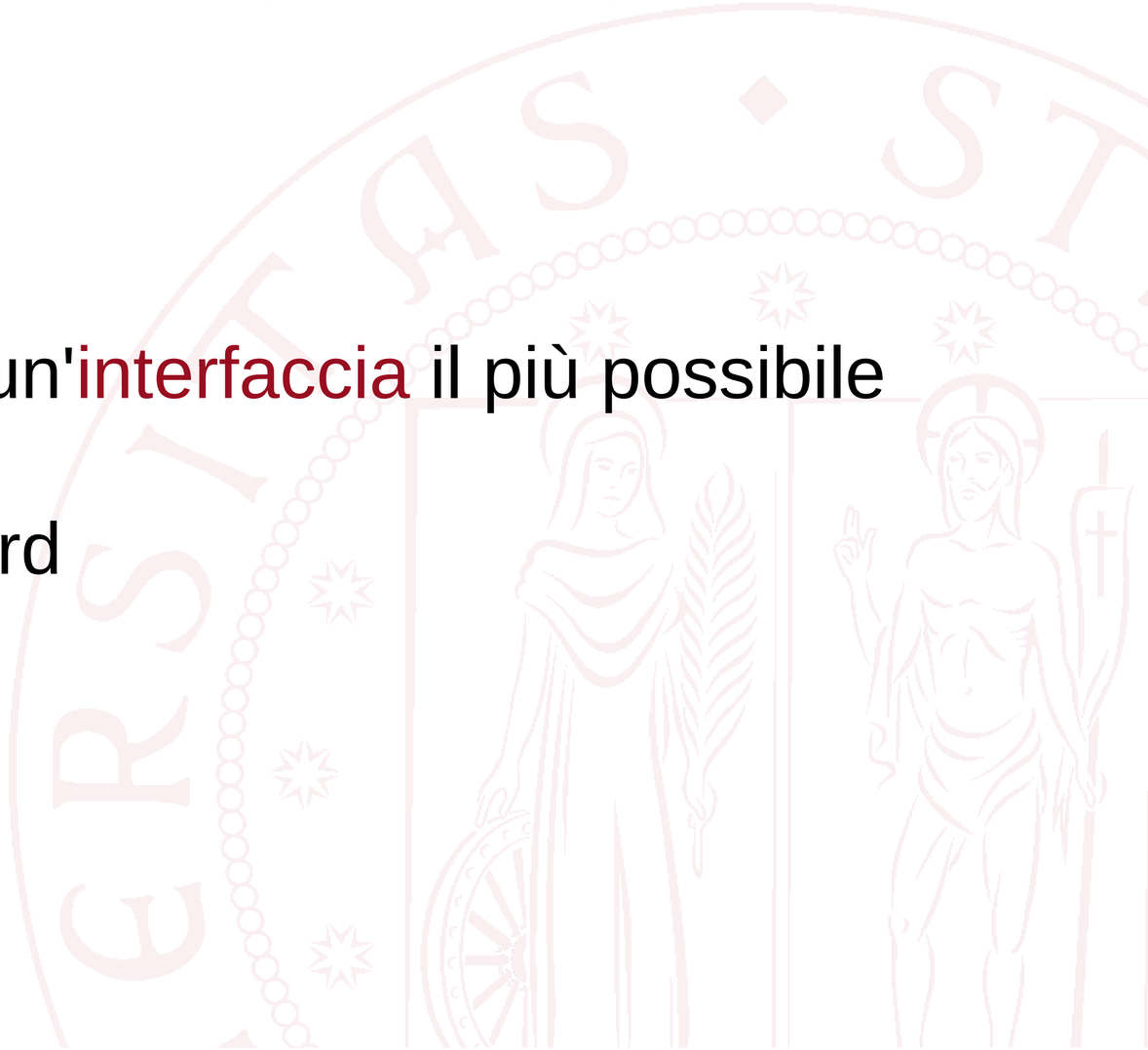
```
p == q                      // True se e solo se p e q puntano
                            // allo stesso elemento

p != q                      // Disequaglianza

*p                          // riferimento all'elemento puntato da p

*p = val                    // scrive nell'elemento puntato da p

val = *p                    // legge dall'elemento puntato da p

++p                         // fa sì che p punti al prossimo elemento
```

Iteratori vs. puntatori

- Concetto di iteratore legato a quello di puntatore
- Infatti un puntatore a un elemento di un array è un iteratore
- Ma molti iteratori non sono puntatori
- Gli iteratori possono definire ad esempio il range-check
- Gli iteratori garantiscono una flessibilità e generalità che i puntatori non hanno

Algoritmi e collezioni di dati

- Per scrivere software di valenza generale è opportuno separare due elementi fondamentali:
  - Collezioni di dati
  - Algoritmi
- Questi elementi interagiscono spesso
- L'interazione deve essere gestita con un'interfaccia il più possibile standard
- STL fornisce questa interfaccia standard



Separare algoritmi e collezioni di dati

- **Disaccoppiamento dell'algoritmo**: non è necessario conoscere i dettagli della struttura dati; è sufficiente usare i relativi iteratori
  - Es: un algoritmo di ordinamento può operare pur senza conoscere i dettagli della struttura dati
- **Disaccoppiamento della struttura dati**: non è necessario esporre tutti i dettagli e gestire tutti i modi d'uso; è sufficiente implementare ed esporre gli iteratori
  - Es: uno `std::vector` non ha necessità di conoscere l'algoritmo che opera su di esso per permettergli di funzionare; è sufficiente che esponga gli iteratori

Esempio di creazione di un iteratore

```
// Restituisce l'elemento in [first, last) che ha il valore più alto

template<typename Iterator>
Iterator high(Iterator first, Iterator last)
{
    Iterator high = first;
    for (Iterator p = first; p != last; ++p)
        if (*high < *p)
            high = p;

    return high;
}
```

Recap

- STL: principi di funzionamento
  - Come risponde all'esigenza di gestire serie di dati in modo generico
- Sequenza
- Iteratori
- Disaccoppiamento di dati e algoritmi