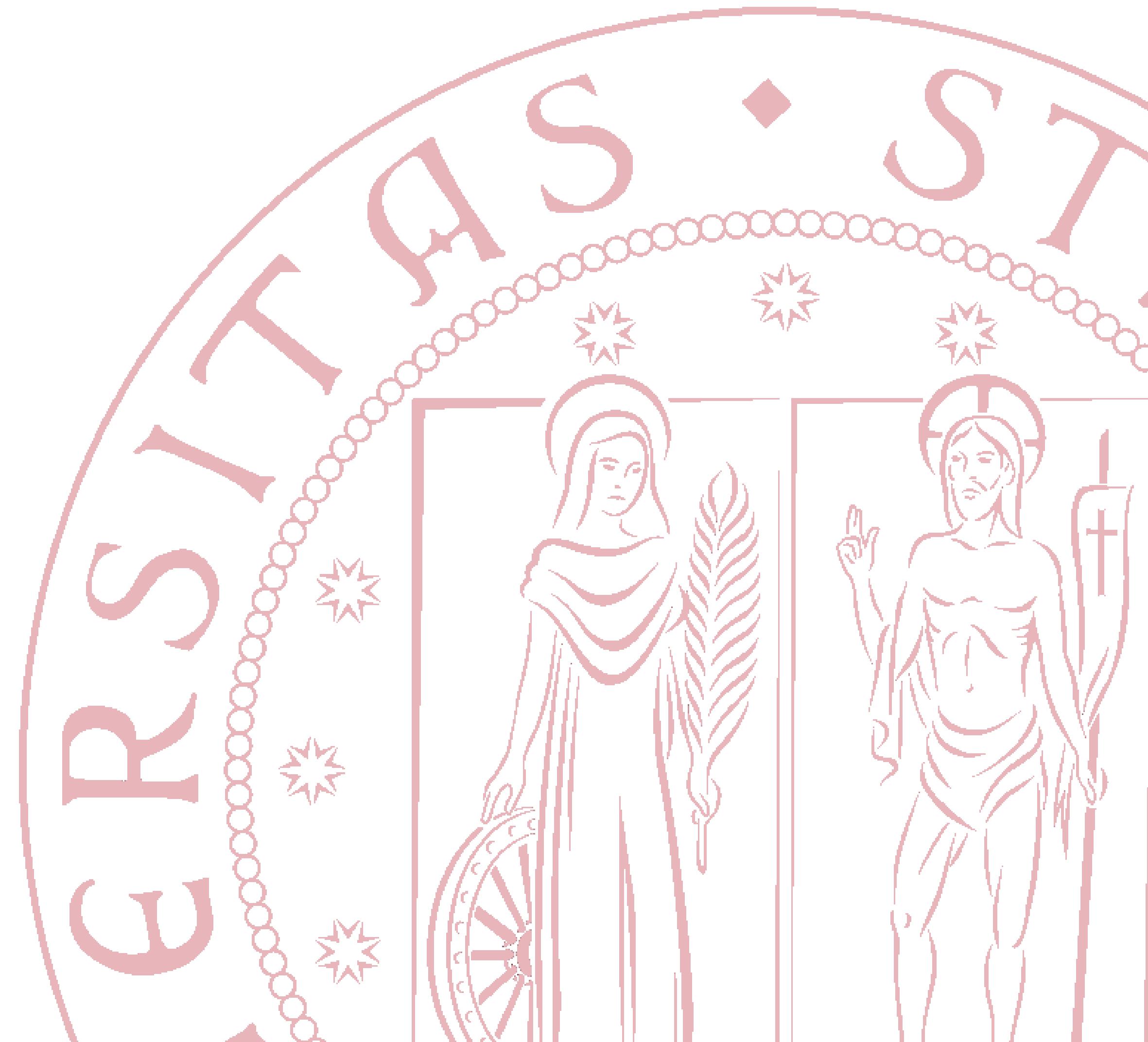


12.3 – Algoritmi STL

Ordinamento e copia

Capitoli:

- 21.4, 21.7, 21.8, 21.9



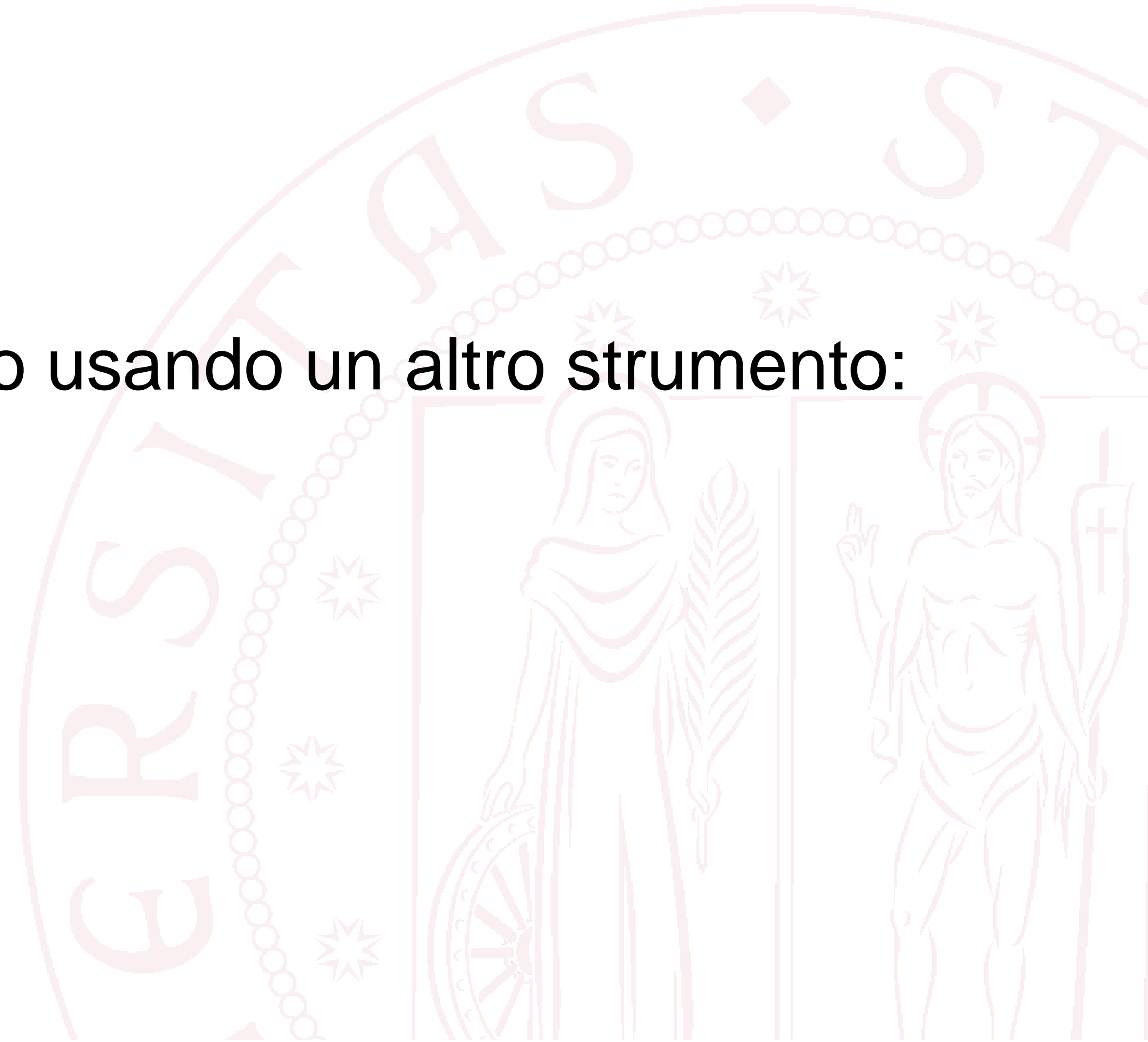
Agenda

- Lambda expression
- Ordinamento
- Criteri di ordinamento
 - Gestiti mediante lambda expression
- Copia
- Copia con predicato



Function object: criticità

- I function object sono una buona tecnica, ma hanno una debolezza:
 - Definiti in una regione di codice
 - Usati altrove
- È possibile rimuovere questo fenomeno usando un altro strumento:
 - Lambda expression



Lambda expression

- Una **lambda expression** (o lambda function) è un modo per:
 - Definire un function object
 - Crearne uno immediatamente e usarlo come argomento di una funzione
- Tale function object è anonimo e non utilizzabile altrove

`std::sort()` e lambda expression



std::sort()

- La funzione std::sort() effettua un ordinamento
- Due versioni:
 - Ordinamento con operator<

```
template<typename Ran>  
    // Ran deve possedere un random_access_iterator  
void sort(Ran first, Ran last);
```

- Ordinamento con funzione dedicata

```
template<typename Ran, typename Cmp>  
    // Ran deve possedere un random_access_iterator  
    // Cmp deve essere una funzione di ordinamento su Ran  
void sort(Ran first, Ran last, Cmp cmp);
```

std::sort()

- std::sort() ordina tra due iteratori
 - Possiamo quindi ordinare anche solo una parte di un container

```
void test(std::vector<int>& v)
{
    // ordina la prima metà
    std::sort(v.begin(), v.begin() + v.size()/2);

    // ordina la seconda metà
    std::sort(v.begin() + v.size()/2, v.end());
}
```

`std::sort()` e container

- Spesso ci interessa ordinare un intero contenitore
- È disponibile la funzione `sort()` per i contenitori
 - Non è parte della STL
 - Definita in `std_lib_facilities.h`

```
template<typename C>
    // C è un contenitore
void sort(C& c)
{
    std::sort(c.begin(), c.end());
}
```


Esempio di ordinamento

- Un esempio con l'algoritmo `std::sort()` e una lambda expression
- Cerchiamo di ordinare un vettore di Record:

```
struct Record {  
    string name;  
    char addr[24];  
    // ...  
};
```

Criteri di ordinamento

- Dato un `std::vector` di `Record`

```
struct Record {  
    string name;  
    char addr[24];  
    // ...  
};
```

```
std::vector<Record> vr;
```

- Abbiamo due modi di ordinarlo – in base a due diversi predicati:

```
struct Cmp_by_name{  
    bool operator()(const Record& a, const Record&b) const  
    { return a.name < b.name; }  
};
```

```
struct Cmp_by_addr{  
    bool operator()(const Record& a, const Record&b) const  
    { return strncmp(a.addr, b.addr, 24) < 0; }  
};
```

Confronto lessicografico

Criteri di ordinamento

- Esistono due criteri di ordinamento perché sono presenti due dati membro
 - Situazione tipica quando si ordina in base ai dati membro

```
// ...  
  
std::sort(vr.begin(), vr.end(), Cmp_by_name());  
  
// ...  
  
std::sort(vr.begin(), vr.end(), Cmp_by_addr());  
  
// ...
```

Criteri di ordinamento

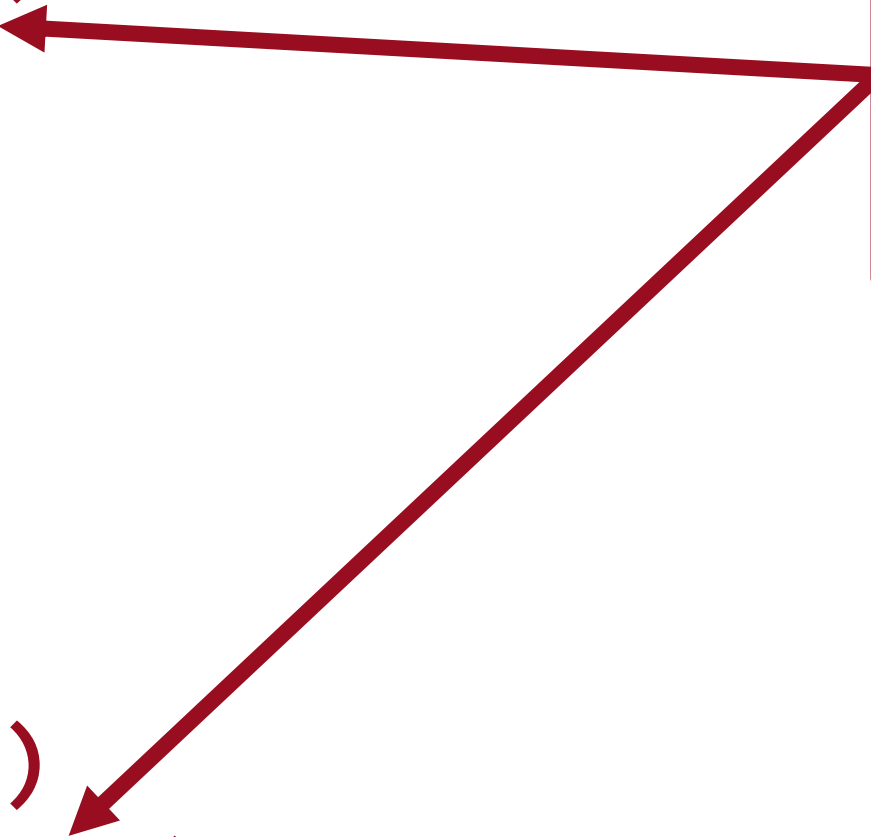
- Esistono due criteri di ordinamento perché sono presenti due dati membro
 - Situazione tipica quando si ordina in base ai dati membro
 - Questa situazione è spesso gestita con le lambda expression
 - Vediamo come



Criteri con lambda expression

```
// ...  
  
std::sort(vr.begin(), vr.end(),  
    [] (const Record& a, const Record& b)  
    { return a.name < b.name; }  
);  
  
// ...  
  
std::sort(vr.begin(), vr.end(),  
    [] (const Record& a, const Record& b)  
    { return strcmp(a.addr, b.addr, 24) < 0; }  
);  
  
// ...
```

all'interno di sort
verrà chiamata questa
espressione



Lambda expression

- Le lambda expression accettano argomenti come le funzioni
- Possono anche **catturare** le variabili locali

```
void func5(std::vector<double>& v, const double& epsilon) {  
  
    std::transform(v.begin(), v.end(), v.begin(),  
        [epsilon](double d) -> double {  
        if (d < epsilon) {  
            return 0;  
        } else {  
            return d;  
        }  
    });  
}
```

Cattura di variabili locali

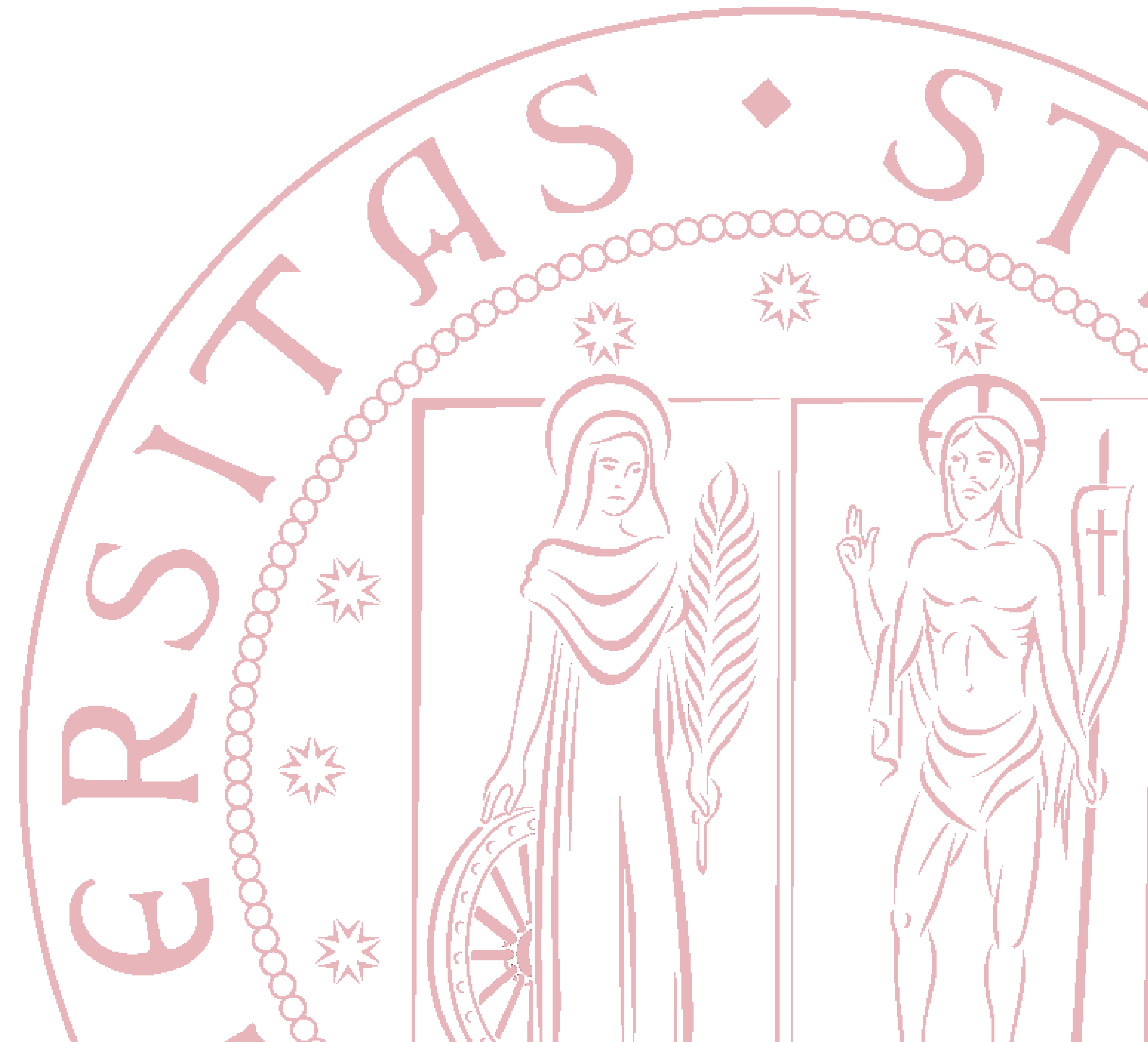
Specifica il tipo ritornato
(opzionale, a meno di
ambiguità)

Lambda expression

- La cattura delle variabili locali rende le lambda expression molto comode

Capture clause	Effetto
[ε, zeta]	Cattura epsilon per riferimento e zeta per valore (copia)
[&]	Cattura tutte le variabili usate nella lambda expression per riferimento
[=]	Cattura tutte le variabili usate nella lambda expression per valore
[&, epsilon]	Cattura tutte le variabili usate nella lambda expression per riferimento, ma epsilon è per valore
[=, ε]	Cattura tutte le variabili usate nella lambda expression per valore, ma epsilon è per riferimento

std::copy()



std::copy()

- Effettua una copia

```
template<typename In, typename Out>
// In: iteratore di input
// Out: iteratore di output
Out copy(In first, In last, Out res)
{
    while(first != last) {
        *res = *first;
        ++res;
        ++first;
    }
    return res;
}
```

`std::copy()`

- Copia di una sequenza in un'altra sequenza
 - Possono essere container diversi
- Spetta all'utente verificare che sia disponibile sufficiente spazio a destinazione
 - Politica simile al range check per i `std::vector`: non sono effettuate operazioni potenzialmente costose e non sempre necessarie

`std::copy()` – utilizzo

- Sorgente: segnalati inizio e fine
- Destinazione: segnalato solo l'inizio

```
void f(std::vector<double>& vd, std::list<int>& li)
{
    if(vd.size() < li.size())
        error("Target container too small\n");
    std::copy(li.begin(), li.end(), vd.begin());
}
```

std::copy_if()

- Esiste anche la copia con verifica di un predicato
 - Stessa sintassi, con un quarto argomento che specifica il predicato

```
void f(const vector<int>& v)
{
    std::vector<int> v2(v.size());
    std::copy_if(v.begin(), v.end(), v2.begin(), Larger_than(6));
}
```

Spunti di approfondimento

- Algoritmi numerici
 - `std::accumulate`
 - `std::inner_product`
 - `std::partial_sum`
 - `std::adjacent_difference`



Recap

- Ordinamento
- Esprimere un criterio di ordinamento con una lambda expression
- Copia
- Gestione dei contenitori sorgente e destinazione per la copia
- Copia subordinata a un predicato

