7.1 – Inizializzazione di oggetti

Libro di testo:

- Capitolo 18.2

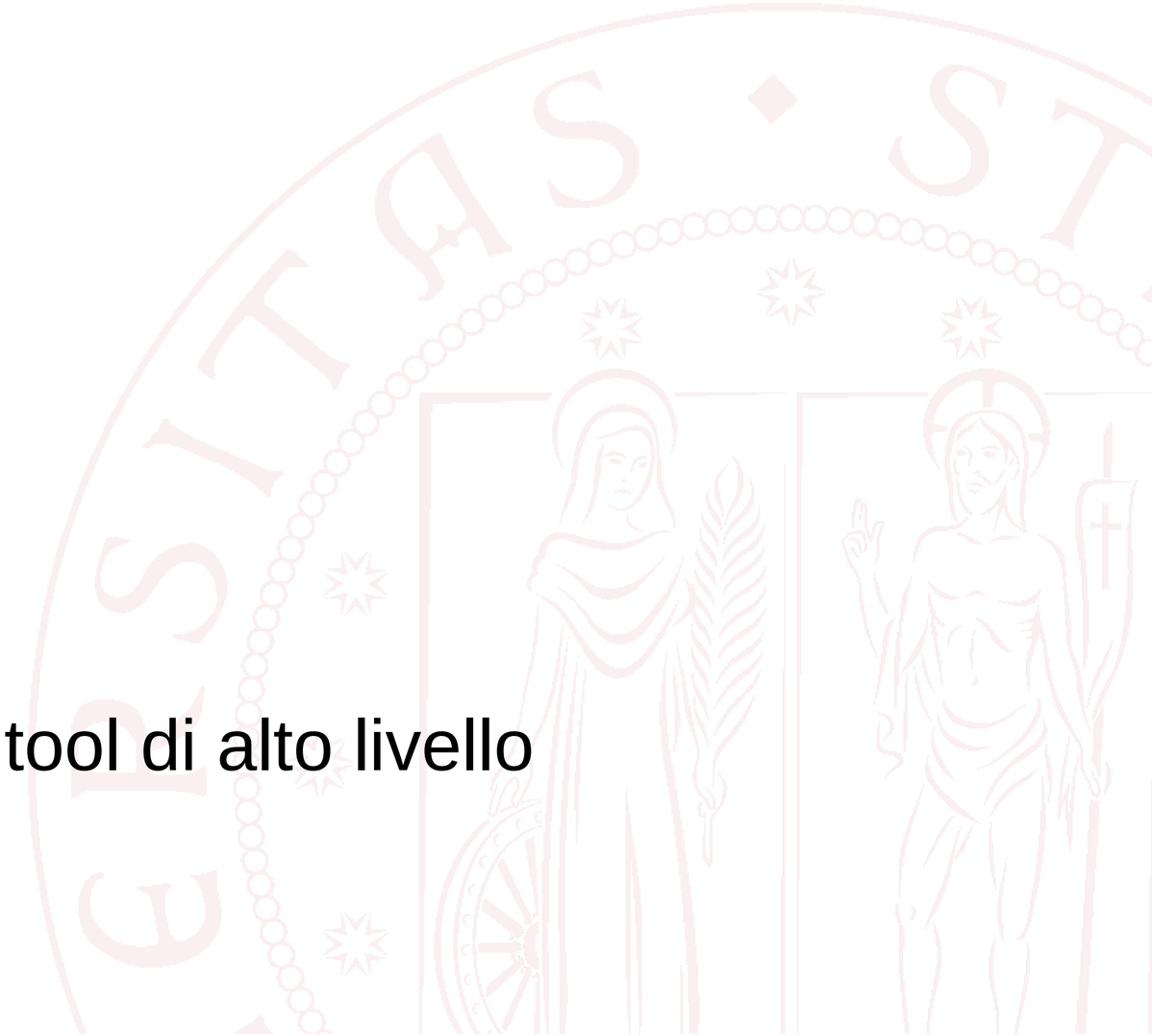
DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Agenda

- Proseguiamo lo sviluppo della nostra classe `vector`
- Inizializzazione
  - `initializer_list`
- Alcuni casi di ambiguità

Da basso ad alto livello (recap)

- A basso livello (es., `array`)
  - Un oggetto ha dimensione fissa
  - Un oggetto ha una posizione fissa
  - Esistono poche operazioni fondamentali
- Ad alto livello (es., `vector`)
  - Comportamento più intelligente
  - Qualcuno deve implementarlo!
- Proseguiamo nell'implementazione di tool di alto livello



Recap

- Riprendiamo il nostro `vector`

```
class vector {
    int sz;
    double* elem;

    public:
        vector(int s = 0) : sz{s}, elem{new double[s]} {
            if(s == 0) elem = nullptr;
        }

        ~vector() {
            delete[] elem;
        }
        void push_back(double d);

        //...
};
```

Costruttore con allocazione dinamica

Check per s=0

Distruttore con deallocazione

Inizializzazione

- Al momento possiamo solo inizializzare usando:

```
vector v3;              // lungo e ripetitivo
v3.push_back(1.2);
v3.push_back(7.89);
v3.push_back(12.34);
```

- Ora, vorremmo farlo nel solito modo:

```
vector v1 = { 1.2, 7.89, 12.34 };   // più compatto!
```

Initializer list

- Una scrittura del tipo:

```
vector v1 = { 1.2, 7.89, 12.34 };    // più compatto!
```

utilizza un oggetto della standard library di tipo `initializer_list<T>`

- Possiamo quindi aggiungere questo costruttore:

```
class vector {
// ...
    vector(initializer_list<double> lst)
        : sz{lst.size()}, elem{new double[sz]}
    {
        std::copy(lst.begin(), lst.end(), elem);
    }
// ...
};
```

Initializer list

```
class vector {
// ...
    vector(initializer_list<double> lst)
        : sz{lst.size()}, elem{new double[sz];}
    {
        std::copy(lst.begin(), lst.end(), elem);
    }
// ...
};
```

`std::copy` algorithm (standard library)

- `std::copy` algorithm: copia una sequenza di elementi delimitata dai suoi primi due argomenti a una sequenza puntata dal terzo argomento
  - **Iteratori** per la definizione della sorgente

Initializer list

- Osserviamo un dettaglio:

```
vector(initializer_list<double> lst)
```

  - È passato per copia!
  - `initializer_list` è usato in questo modo, come richiesto dal linguaggio
  - `initializer_list` è semplicemente un handle a elementi allocati "altrove"

Casi di ambiguità

- Le seguenti sono entrambe corrette

```
vector v1 {3};
vector v2(3);
```

- Qual è la differenza?

```
vector v1 {3};
vector v2(3);
```

- Qual è la differenza?

```
vector v11 = {1, 2, 3};
vector v12 {1, 2, 3};
```