4.1 – Tipi definiti dall'utente

Introduzione

Libro di testo:

- Capitolo 9.1, 9.2, 9.3

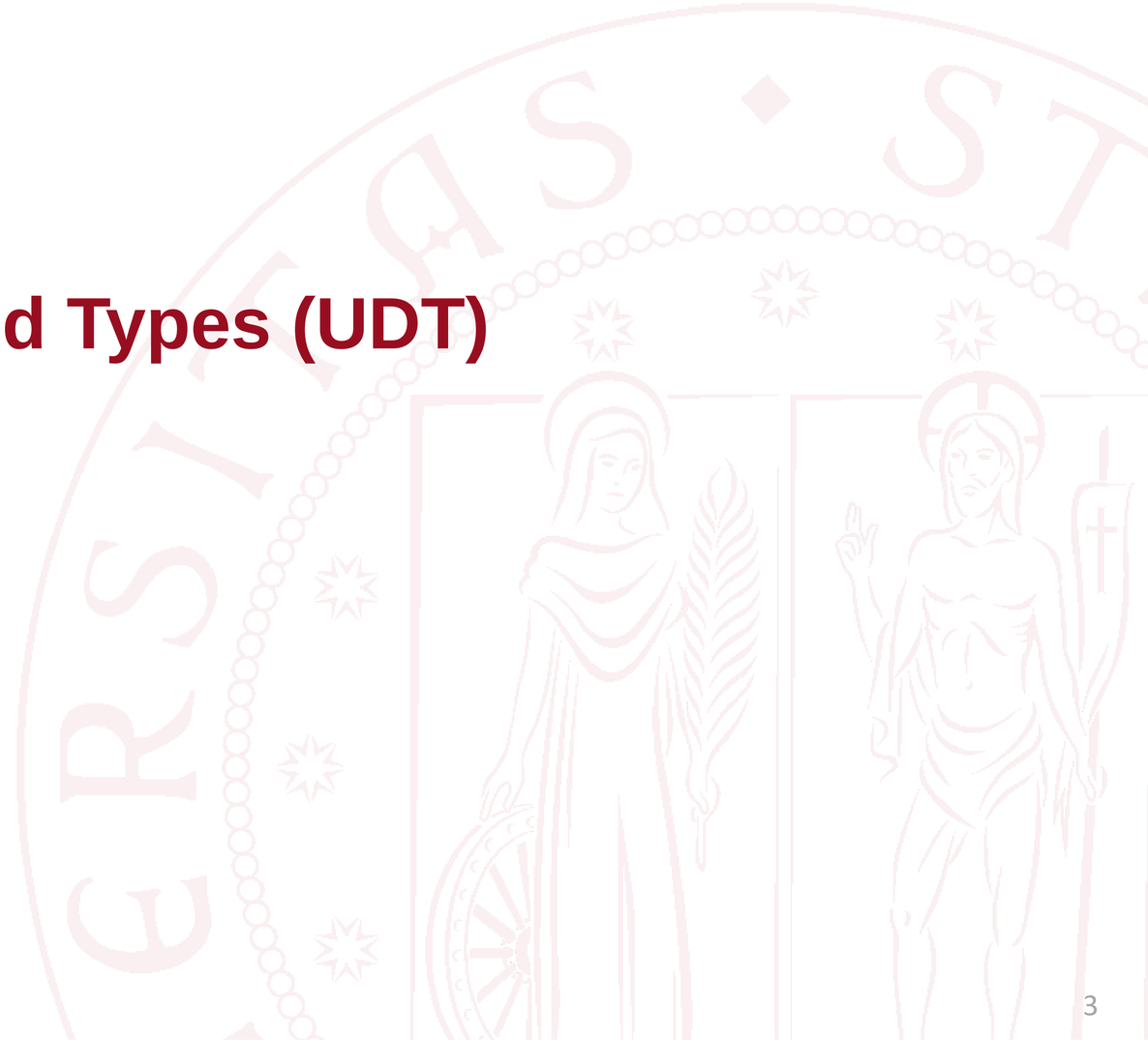
DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Agenda

- Introduzione alle classi
- Accesso ai dati
- Interfaccia e implementazione

Tipi

- In C++ forte focus sui tipi
- **Tipi built-in**
  - `int`
  - `char`
  - `double`
  - …
- **Tipi definiti dall'utente / User Defined Types (UDT)**
  - Classi definite in `std`
    - `string`
    - `vector`
    - `iostream`
  - Classi definite dal programmatore
  - Struct definite dal programmatore



Libreria `std`

- `std::vector`, `std::string`, `std::list`, `std::ostream`, …

  Sono considerati UDT

  - Costruiti con le stesse tecniche delle classi definite dal programmatore
    - È possibile costruire i contenitori con gli strumenti messi a disposizione dal C++
  - Sono comunque parte del linguaggio come i tipi built-in
  - I programmatori della libreria standard non hanno "superpoteri"

Perché le classi?

- La programmazione orientata agli oggetti porta alla creazione di nuovi tipi
- "Types are good for directly representing ideas in code" (BS)
- Due elementi chiave:
  - **Rappresentazione** (representation): un tipo deve sapere come rappresentare i dati necessari in un oggetto
    - Concetto di valore corrente o di stato
  - **Operazioni** (operations): un tipo deve sapere che operazioni possono essere applicate a un oggetto (cioè ai dati che lo rappresentano)

Stato

- Quasi tutte le macchine hanno uno stato
- Che stato potremmo definire nei seguenti casi?
  - Macchinetta del caffè
  - Tazza di caffè (!!)
  - Motore di un'auto
    - Auto a combustione
    - Auto elettrica

Strumenti per definire tipi

- C++ fornisce due tipi di UDT: **classe** e **enumerazione**

- Una classe
  - Rappresenta direttamente un concetto in un programma
  - Specifica, per ogni tipo:
    - Come sono rappresentati i relativi oggetti
      - Attenzione al significato di *oggetto*!
    - Come questi oggetti possono essere creati
    - Come questi oggetti possono essere usati
    - Come questi oggetti possono essere distrutti

  - Se pensiamo a qualcosa come a un'entità separata, probabilmente può essere mappata in una classe

Classi

Classi

- Una classe è composta da:
  - Tipi built-in
  - Altri UDT
  - Funzioni
- Le entità utilizzate per definire la classe sono chiamati **membri**
  - **Data member** (dato membro)
  - **Function member** (funzione membro)

Classi

Un esempio semplice

```
class X {
    public:
        int m;                          // data member
        int mf(int v) {                 // function member
            int old = m;
            m = v;
            return old;
        }
};
```

Accesso ai membri

- È possibile accedere ai membri con la notazione:

oggetto.membro

```
class X {
  public:
      int m;                          // data member

      int mf(int v) {                 // function member
        int old = m;
        m = v;
        return old;
    }
};
```

```
X var;

var.m = 7;

int x = var.mf(9);
```

Interfaccia e implementazione

- Paradigma già introdotto
- Naturale implementazione con le classi
- Interfaccia (**`public`**)
  - Punto di vista dell'utente
  - Accessibile agli utenti
- Implementazione (**`private`**)
  - Punto di vista del programmatore
  - Non direttamente accessibile
  - Lettura/scrittura solo tramite l'interfaccia
    - Permette il controllo dello stato (es., coerenza)

- Di default i membri di una classe sono privati

Public & private

```
class X {
  public:
       // membri pubblici:
       // interfaccia utente
       // funzioni
       // tipi
       // eventuali dati (meglio tenerli privati)

  private:
       // membri privati
       // dettagli implementativi
       // funzioni
       // tipi
       // dati
};
```

Public & private

```
class X {
    private:
      int m;                          // data member

    public:
      int mf(int v) {                 // function member
        int old = m;
        m = v;
        return old;
       }
};
```

```
X var;

var.m = 7;              // errore!

int x = var.mf(9);      // ok!
```

`struct`

- Le `struct` sono varianti delle classi
- Unica differenza tecnica: i membri di default sono pubblici
- Solitamente usate quando ci sono dati ma non funzioni
- Eredità del C

```
struct X {

    int m;

};
```

=

```
class X {

    public:
        int m;
};
```

Progettazione di una classe

- Il C++ supporta la programmazione a oggetti con una serie di strumenti potenti
  - Funzioni membro
  - Costruttori
  - Protezione dei dati
  - Overloading degli operatori
  - Gestione della copia dei dati
  - (Helper function)

Recap

- UDT
- Classi
  - Stato
  - Interfaccia / implementazione: public/private
- Strumenti del C++ a supporto della programmazione a oggetti