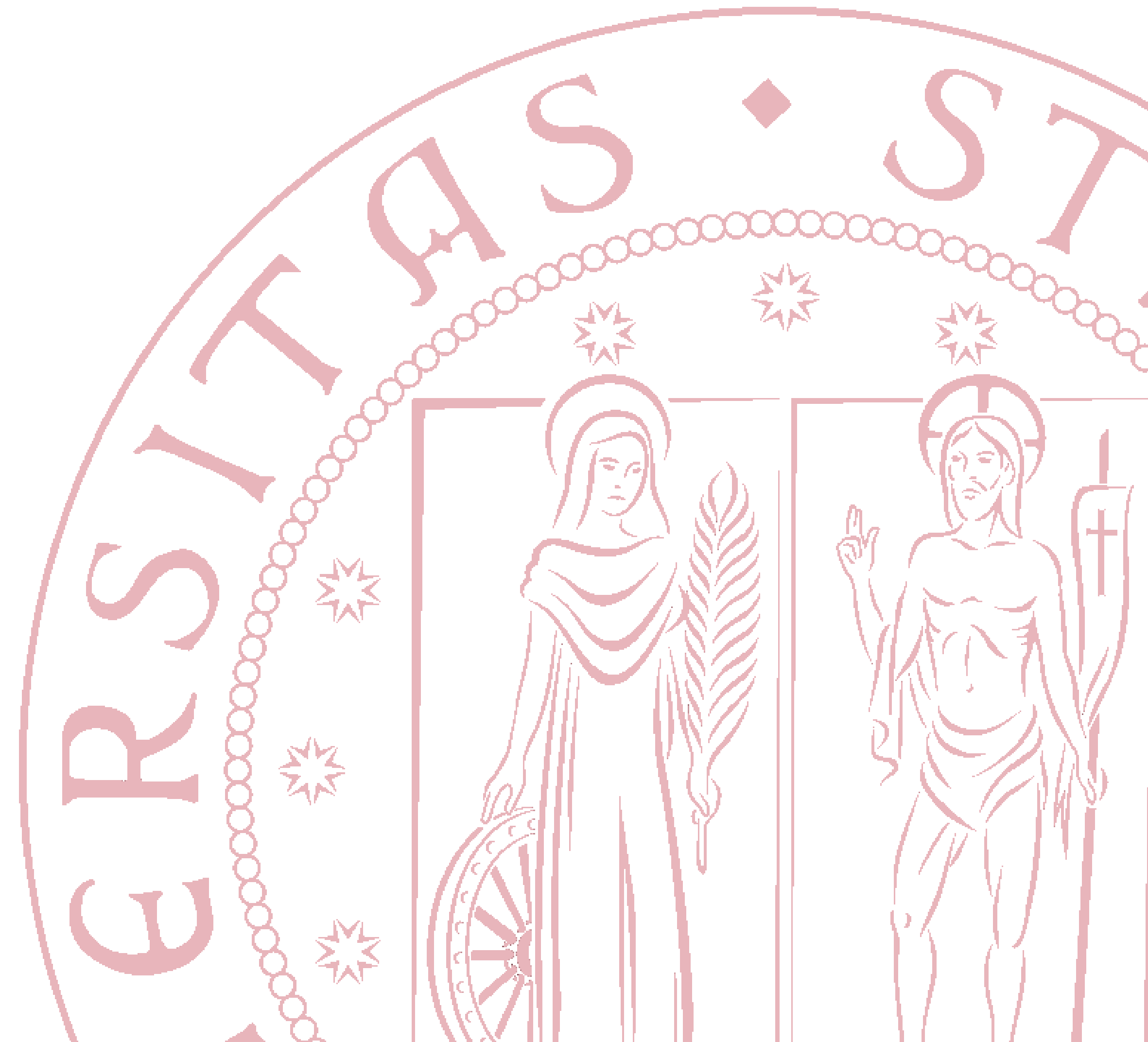


2.1 – Variabili e type safety

Libro di testo:

- Capitolo 3



Agenda

- Definizioni: tipo, oggetto, valori, variabile
- Scrivere su una variabile
- Naming
- Type safety e conversioni



Tipo, oggetto, variabile, valore

- **Tipo:** definisce una tipologia di contenuto, un range di valori e un insieme di operazioni per un oggetto
esempi?
- **Oggetto:** una regione di memoria con un tipo che specifica quale tipo di dato può essere inserito
di che tipo memoria stiamo parlando?
- **Variabile:** un oggetto con un nome (named object)
- **Valore:** l'elemento posto dentro alla variabile
 - Insieme di bit in memoria sono interpretati *in funzione del tipo*

```
int age = 42;
```

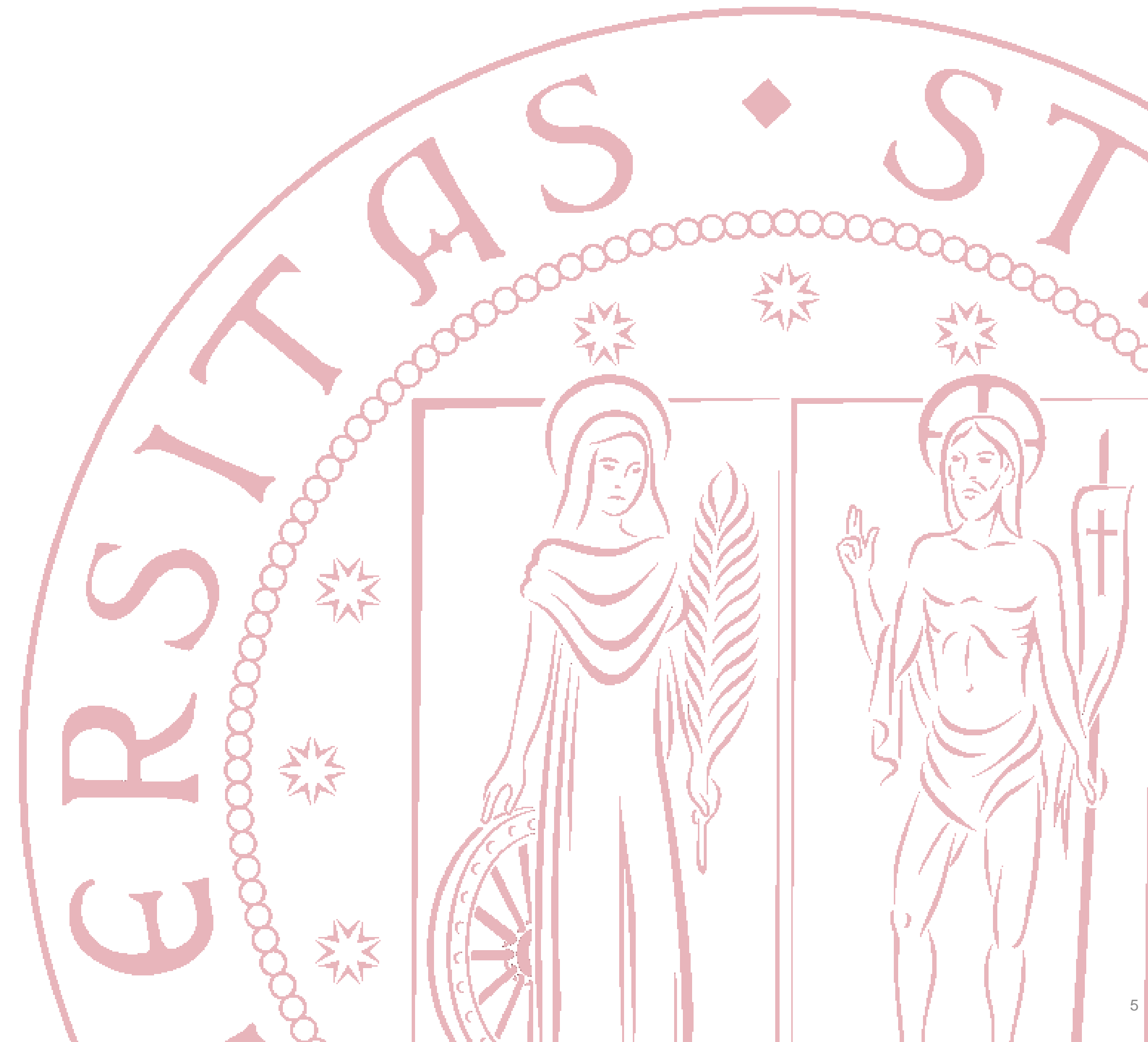
int:
age: 42

Tipo di una variabile

- Il tipo di una variabile determina le operazioni che possono essere effettuate e il loro significato
- Ogni tipo ha un suo **formato literal**:

```
39          // int
3.5         // double
'.'         // char
"Annemarie" // string
true        // bool
```

Scrivere su una variabile



Assegnamento e inizializzazione

- L'operatore più usato è l'assegnamento (=)
- Lo stesso simbolo è usato per due operazioni simili ma concettualmente diverse

Inizializzazione

```
int a = 3;
```

Assegnamento

```
a = 4;
```

Qual è la differenza?

Assegnamento composto

```
a += 7;  
b -= 9;  
c *= 2;
```

Type safety

- Il tipo di una variabile condiziona:
 - La natura dei dati contenuti
 - La dimensione in byte
- Il passaggio di dati tra variabili dello stesso tipo è concettualmente semplice
- Che succede se cerchiamo di trasferire dati tra variabili di tipi diversi?
 - **Problema del type safety**

Type safety

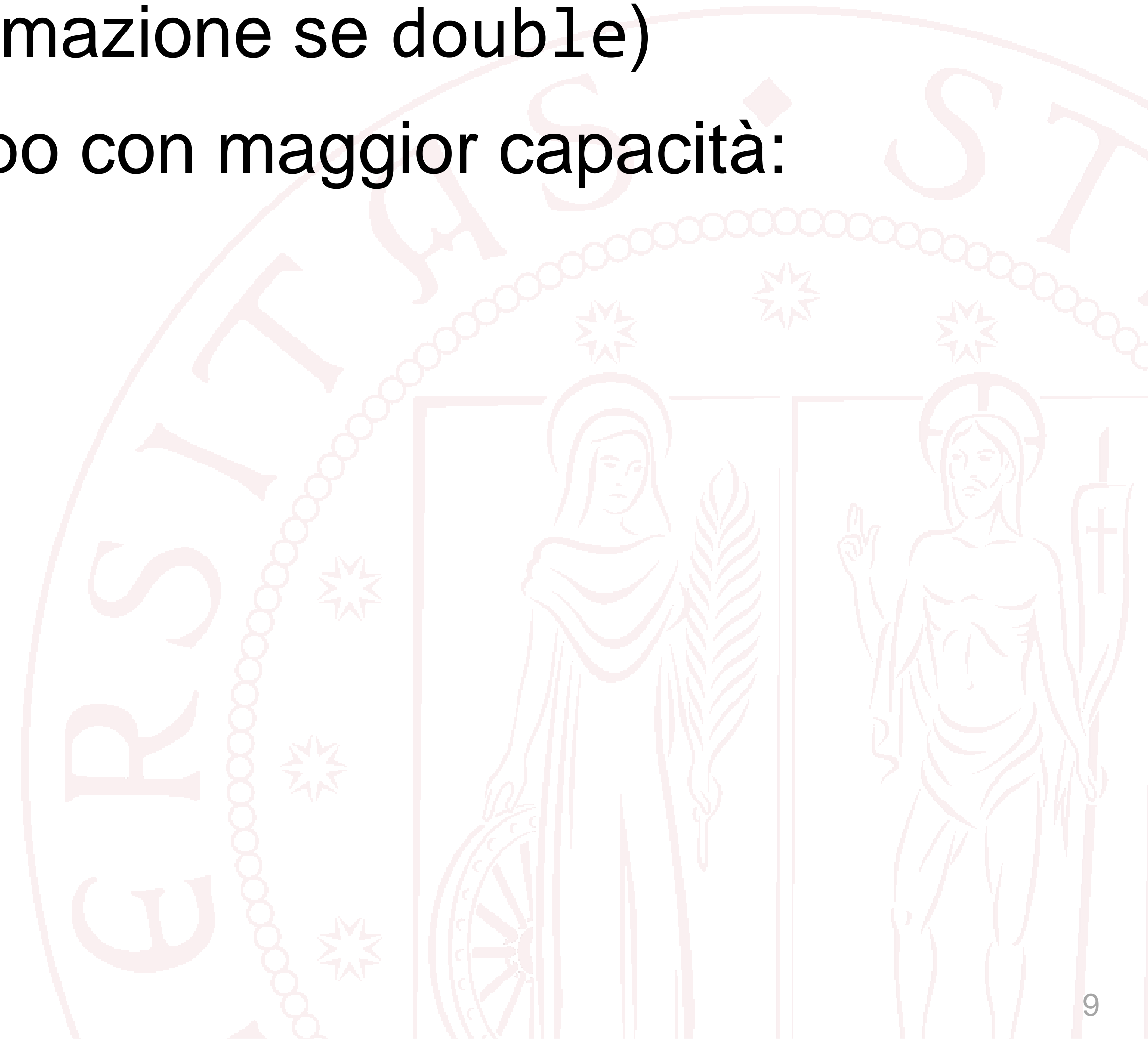
- Alcune operazioni non sono type-safe
- Es: usare una variabile non inizializzata

```
double x;           // valore di x indefinito  
double y = x;       // valore di y indefinito
```

- È possibile un errore HW se x è usata senza inizializzazione
- **Inizializzate sempre le variabili!**
- Il C++ è fortemente tipizzato, e la type safety deve essere sempre garantita

Conversioni sicure

- Le conversioni sicure preservano il valore (o lo convertono nella miglior approssimazione se double)
- Sono sicure le conversioni verso un tipo con maggior capacità:
 - `bool` \rightarrow `char`
 - `bool` \rightarrow `int`
 - `bool` \rightarrow `double`
 - `char` \rightarrow `int`
 - `char` \rightarrow `double`
 - `int` \rightarrow `double`



Conversioni sicure

- Le conversioni `int` \rightarrow `double` permettono di usare espressioni che contengono entrambi i tipi

```
double d1 = 2.3;  
double d2 = d1 + 2; // 2 convertito in 2.0
```

Conversioni non sicure

- Le conversioni non sicure non garantiscono che il valore convertito sia equivalente a quello di partenza ("**narrowing**")
- double → int
- double → char
- double → bool
- int → char
- int → bool
- char → bool
- sizeof(bool) → 1 byte
- sizeof(char) → 1 byte
- sizeof(int) → 4 bytes
- sizeof(float) → 4 bytes
- sizeof(double) → 8 bytes
- **Sono raramente rilevate dal compilatore!!!**

Conversioni non sicure

Qual è l'output di questo programma? Perché?

```
int main(void)
{
    int a = 20000;
    char c = a;
    int b = c;
    if (a != b)
        std::cout << " a e b sono diversi!\n";
    else
        std::cout << "a e b sono uguali!\n";

    return 0;
}
```

Controllo su inizializzazione

- Le conversioni non sicure sono accettate per eredità storiche (da C)
- C++11 introduce una notazione di inizializzazione che **evita le narrowing conversions**:

{ } – universal and uniform initialization

- Il compilatore può controllare i literal

```
double x {2.7}; // OK
int y {x};      // errore: narrowing conversion

int a {1000};   // OK
char b {a};     // errore: narrowing conversion

char b1 {1000}; // errore: valore fuori limiti
char b2 {48};   // OK
```

Conversione di tipi

- Due modi per convertire:
 - Tipo {valore} – con controllo di narrowing
 - Tipo (valore) – senza controllo

```
double d = 2.5;  
int i = 2;
```

```
double d2 = d/i; // d2 == 1.25
```

```
int i2 = d/i;    // i2 == 1
```

```
int i3 {d/i};    // errore: double -> int, narrowing
```

Dare nomi alle variabili



Nomi per le variabili

- I nomi sono importanti per gli umani!
- Significativi ed evocativi
 - Ma non troppo lunghi
- Evitare acronimi: `mtbf`, `TLA`, `myw`, `nbv` possono confondere
- Sono accettabili nomi brevi solo se definiti in una convenzione o usati in ambito molto ristretto
 - Tipico indice del loop: `i` (ok!)
- Nomi eccessivamente lunghi sono problematici
 - `element_count` meglio di `the_number_of_elements`

Convenzione per i nomi

- È importante scegliere una convenzione e rispettarla
 - Rende molto più semplice la lettura del codice
- Convenzione suggerita: Google C++ Style Guide
 - <https://google.github.io/styleguide/cppguide.html#Naming>