

# Corso “Programmazione 1”

## Capitolo 05: Le Funzioni

Docente: **Marco Roveri** - `marco.roveri@unitn.it`  
Esercitori: **Giovanni De Toni** - `giovanni.detoni@unitn.it`  
**Stefano Berlato** - `stefano.berlato-1@unitn.it`  
C.D.L.: Informatica (INF)  
A.A.: 2021-2022  
Luogo: DISI, Università di Trento  
URL: <https://bit.ly/2VgfYwJ>



Ultimo aggiornamento: 13 ottobre 2021

# Terms of Use and Copyright

## USE

This material (including video recording) is intended solely for students of the University of Trento registered to the relevant course for the Academic Year 2021-2022.

## SELF-STORAGE

Self-storage is permitted only for the students involved in the relevant courses of the University of Trento and only as long as they are registered students. Upon the completion of the studies or their abandonment, the material has to be deleted from all storage systems of the student.

## COPYRIGHT

The copyright of all the material is held by the authors. Copying, editing, translation, storage, processing or forwarding of content in databases or other electronic media and systems without written consent of the copyright holders is forbidden. The selling of (parts) of this material is forbidden. Presentation of the material to students not involved in the course is forbidden. The unauthorised reproduction or distribution of individual content or the entire material is not permitted and is punishable by law.

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

# Funzioni ricorsive

- In C++ una funzione può invocare se stessa (**funzione ricorsiva**)
- ... o due o più funzioni possono chiamarsi a vicenda (**funzioni mutualmente ricorsive**)
- Formulare alcuni problemi in maniera ricorsiva risulta naturale:
  - il fattoriale:  $0! \stackrel{\text{def}}{=} 1; n! \stackrel{\text{def}}{=} n \cdot (n-1)!$
  - pari/dispari:  $\text{even}(n) \iff \text{odd}(n-1); \text{odd}(n) \iff \text{even}(n-1);$
  - espressioni:  $\text{somma} \stackrel{\text{def}}{=} \text{numero}; \text{somma} \stackrel{\text{def}}{=} (\text{somma} + \text{somma})$
- Due componenti:
  - una o più **condizioni di terminazione**
  - una o più **chiamate ricorsive**
- **Intrinseco rischio di produrre sequenze infinite**
  - Analoghe considerazioni rispetto ai cicli
- **Alcune “insidie” computazionali**
  - Es: funzione di Fibonacci:  $f_0 \stackrel{\text{def}}{=} 1; f_1 \stackrel{\text{def}}{=} 1; f_n \stackrel{\text{def}}{=} f_{n-1} + f_{n-2}$

Ricorsione fortemente collegata al **principio di induzione** matematico.

# Esempi

- **fattoriale:**  
{ FUNZIONI\_RICORSIVE/fact\_nocomment.cc }
- **..., con chiamate tracciate:**  
{ FUNZIONI\_RICORSIVE/fact.cc }
- **..., errore (loop infinito) :**  
{ FUNZIONI\_RICORSIVE/fact\_infloop.cc }
- **..., stack tracciato :**  
{ FUNZIONI\_RICORSIVE/fact\_stack.cc }
- **funzioni mutualmente ricorsive:**  
{ FUNZIONI\_RICORSIVE/pariDispari.cc }
- **Fibonacci:**  
{ FUNZIONI\_RICORSIVE/fibonacci\_nocomment.cc }
- **..., con chiamate tracciate:**  
{ FUNZIONI\_RICORSIVE/fibonacci.cc }
- **versione iterativa:**  
{ FUNZIONI\_RICORSIVE/fibonacci\_iterativa.cc }

# Nota sulla ricorsione

La realizzazione ricorsiva di una funzione può richiedere due funzioni:

- una funzione ausiliaria ricorsiva, con un **parametro di ricorsione** aggiuntivo (simile a contatore in loop)
  - una funzione principale (**wrapper**) che chiama la funzione ricorsiva con un valore base del parametro di ricorsione
  - situazione molto frequente nell'uso di array (prossimo capitolo)
- 
- **Esempio di funz. ricorsiva che necessita wrapper:**  
`{ FUNZIONI_RICORSIVE/stampanumeri.cc }`
  - **... variante 1:**  
`{ FUNZIONI_RICORSIVE/stampanumeri1.cc }`
  - **... variante 2:**  
`{ FUNZIONI_RICORSIVE/stampanumeri2.cc }`
  - **analoga variante del fattoriale, con wrapper:**  
`{ FUNZIONI_RICORSIVE/fact_rec1.cc }`

# Ricorsione vs. Iterazione

- Ricorsione spesso più naturale, semplice ed elegante
- Efficienza della ricorsione critica:
  - Attenzione a chiamate identiche in rami diversi! (es. Fibonacci)  
⇒ rischio esplosione combinatoria
  - Dimensione dello stack dipende dalla profondità di ricorsione  
⇒ notevole overhead e spreco di memoria  
⇒ **quando possibile, tipicamente iterazione più efficiente**  
⇒ passando oggetti “grossi”, **è indispensabile usare passaggio per riferimento o puntatore**
- Molte funzioni ricorsive possono essere riscritte in forma iterativa:
  - **tail recursion**: una chiamata ricorsiva, operata **come ultimo passo**
    - Es: somma, pari/dispari, ...
  - in generale, quando non comporta una “biforcazione”
    - Es: fattoriale, Fibonacci, ...
  - g++ -O2 effettua una conversione da tail-recursive in iterative

## Da ricorsione in coda a iterazione (caso void)

<pre><b>void</b> F(<b>int</b> x,...) {   <b>if</b> (CasoBase(x,...))     {IstrBase(...);}   <b>else</b> {     Istr(...);     x=agg(x,...);     F(x,...);   } }</pre>	$\iff$	<pre><b>void</b> F(<b>int</b> x,...) {   <b>while</b> (!CasoBase(x,...)) {     Istr(...);     x=agg(x,...);   }   {IstrBase(...);} }</pre>
--	--------	--

- Esempio funzione void tail-recursive :  
  { FUNZIONI\_RICORSIVE/stampanumeri3.cc }
- ... corrispondente versione iterativa :  
  { FUNZIONI\_RICORSIVE/stampanumeri3\_while.cc }

## Da ricorsione in coda a iterazione (caso generale)

```
type F(int x,...) {  
  if (CasoBase(x,...))  
    res = IstrBase(...);  
  else {  
    Istr(...);  
    x=agg(x,...);  
    res = F(x,...);  
  }  
  return res;  
}  
  
type F(int x,...) {  
  while (!CasoBase(x,...)) {  
    Istr(...);  
    x=agg(x,...);  
  }  
  res = IstrBase(...);  
  return res;  
}
```



- Esempio funzione tail-recursive:  
{ FUNZIONI\_RICORSIVE/sum.cc }
- ... corrispondente versione iterativa:  
{ FUNZIONI\_RICORSIVE/sum\_while.cc }
- Compilazione di funzioni tail-recursive in iterative:  
{ FUNZIONI\_RICORSIVE/tailrecursive-comp.cc }



# Ricorsione vs. Iterazione II

- ...
- Talvolta **non** è agevole riscrivere la ricorsione in forma iterativa
  - funzioni non-tail recursive, chiamate multiple
  - Es: manipolazione di espressioni

Esempio di gestione di espressioni:

{ FUNZIONI\_RICORSIVE/espressione.cc }

- In generale, convertire una funzione ricorsiva in iterativa richiede l'uso di uno stack

Vedere file `ESERCIZI_PROPOSTI.txt`