Programmazione e Strutture Dati (PR&SD)
I° ANNO – Informatica
Prof. V. Fuccella

RICORSIONE

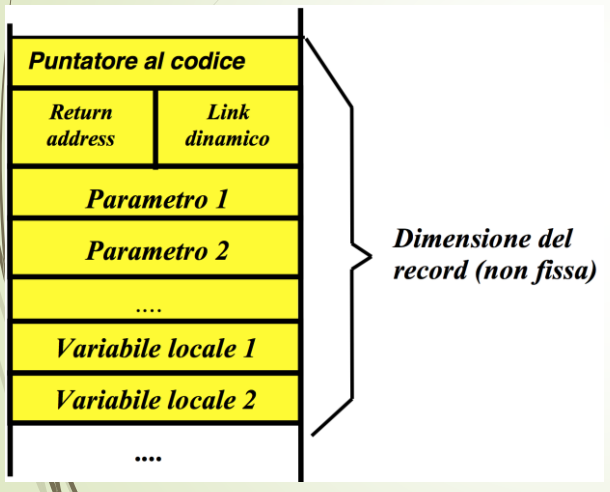
Agenda

- Record di attivazione
- Ricorsione
- Iterazione VS Ricorsione
- Ricorsione Tail
- Strutture Ricorsive

Record di attivazione

- Ogni volta che viene invocata una funzione viene creata dinamicamente una struttura dati detta RECORD DI ATTIVAZIONE.
  - si crea di una nuova attivazione (istanza) della funzione chiamata
  - viene allocata la memoria per i parametri e per le variabili locali
  - si effettua il passaggio dei parametri
  - si trasferisce il controllo alla funzione chiamata
  - si esegue il codice della funzione

Record di attivazione



- È il "mondo della funzione": contiene:
  - i parametri formali
  - le variabili locali
  - l'indirizzo di ritorno (Return address RA) che indica il punto a cui tornare (nel codice del chiamante) al termine della funzione.
  - un collegamento al record di attivazione del chiamante (Link Dinamico DL)
  - l'indirizzo del codice della funzione (puntatore alla prima istruzione del corpo)

5

Record di attivazione

Ciclo di Vita

- Il record di attivazione associato a una chiamata di una funzione f:
  - È creato al momento della invocazione di f
  - Permane per tutto il tempo in cui f è in esecuzione
  - È distrutto (deallocato) al termine dell'esecuzione
- La dimensione del record di attivazione
  - varia da una funzione all'altra
  - per una data funzione, è fissa e calcolabile a priori

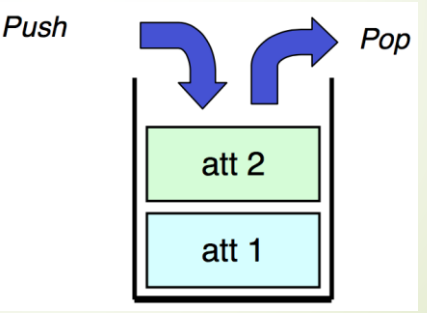
6

Stack

- L'area di memoria in cui vengono allocati i record di attivazione viene gestita come una **lista** LIFO
  - nella quale ogni elemento è un record di attivazione.
- La gestione dello stack avviene mediante due operazioni:
  - **push:** aggiunta di un elemento (in cima alla pila)
  - **pop:** prelievo di un elemento (dalla cima della pila)

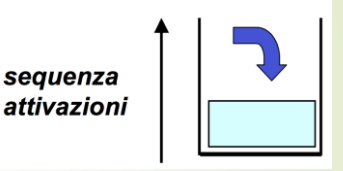
Stack

- L'ordine di collocazione dei record di attivazione nello stack indica la cronologia delle chiamate:

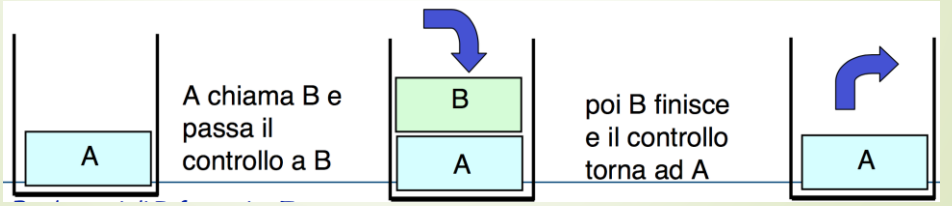


Stack

- Normalmente lo STACK dei record di attivazione si disegna nel modo seguente:



- Quindi, se la funzione A chiama la funzione B, lo stack evolve nel modo seguente



9

Stack
Chiamate annidate

Programma:

- `int R(int a) { return a+1; }`
- `int Q(int x) { return R(x); }`
- `int P(void) { int a=10; return Q(a); }`
- `main() { int x = P(); }`

Sequenza chiamate:

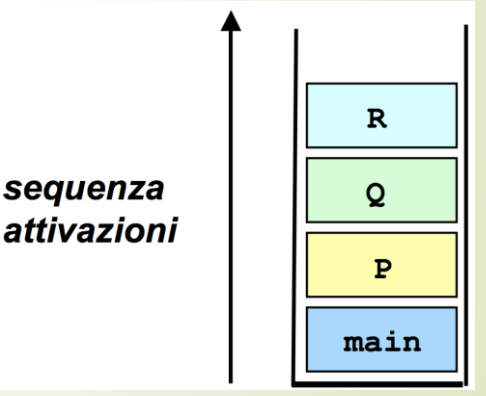
- `S.O.` → `main` → `P()` → `Q()` → `R()`

10

Stack
Chiamate annidate

Sequenza chiamate:

- `S.O.` → `main` → `P()` → `Q()` → `R()`



Agenda

- Record di attivazione
- Ricorsione
- Iterazione VS Ricorsione
- Ricorsione Tail
- Strutture Ricorsive

Programmazione Ricorsiva

- Un sottoprogramma ricorsivo è:
  - un sottoprogramma che richiama direttamente o indirettamente se stesso.
- I linguaggi che gestiscono la ricorsione, lo fanno mediante record di attivazione
- Operativamente, risolvere un problema con un approccio ricorsivo comporta
  - Identificare un "caso base", con soluzione nota
  - esprimere la soluzione al caso generico *n* in termini dello *stesso problema in uno o più casi più semplici* (n-1, n-2, etc.)

Funzioni Matematiche Ricorsive

- Una funzione matematica è definita ***ricorsivamente*** quando nella sua definizione compare un riferimento a se stessa
- È basata sul *principio di induzione* matematica:

  – se una proprietà P vale per $n=n_0$ (**CASO BASE**)

  – e si può provare che, ***assumendola valida per n***, allora vale per n+1
  - allora P vale per ogni $n \geq n_0$

Funzioni Ricorsive
Il Fattoriale

- La funzione fattoriale(n), denotato come n!, è definito per tutti gli interi n>=0 come:
  - n! = 1 se n = 0
  - n! = n * (n-1)! se n > 0

15

La Ricorsione

Il Fattoriale

- Servitore (funzione chiamata) & Cliente (chiamante): **fact** è sia servitore che cliente:

```
int fact(int n)
{
  if (n==0) return 1;
  else return n*fact(n-1);
}

main() {
  int fz,z = 5;
  fz = fact(z-2);
}
```

Il `main` chiama `fact(3)`

`fact(3)` chiama `fact(2)`

`fact(2)` chiama `fact(1)`

`fact(1)` chiama `fact(0)`

16

La Ricorsione

Il Fattoriale

- Servitore (funzione chiamata) & Cliente (chiamante): **fact** è sia servitore che cliente:

```
int fact(int n)
{
  if (n==0) return 1;
  else return n*fact(n-1);
}

main() {
  int fz,z = 5;
  fz = fact(z-2);
}
```

`fact(0)` restutisce 1

`fact(1)` restituisce 1

`fact(2)` restituisce 2

`fact(3)` restituisce 6

La Ricorsione

Il Fattoriale

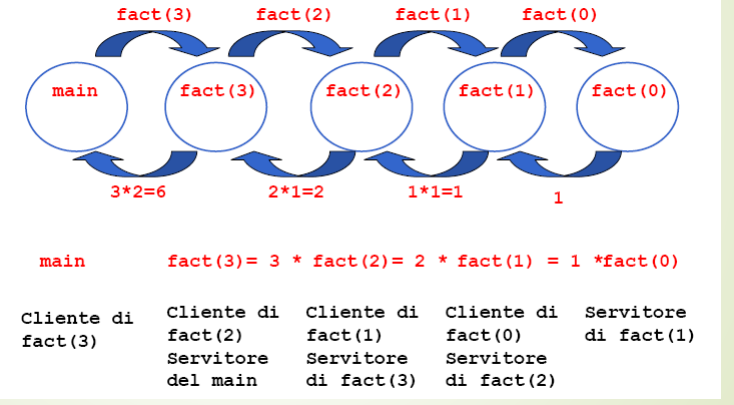
- Servitore (funzione chiamata) & Cliente (chiamante): **fact** è sia servitore che cliente:

```
int fact(int n)
{
  if (n==0) return 1;
  else return n*fact(n-1);
}

main() {
  int fz,z = 5;
  fz = fact(z-2);
}
```

Il `main` riceve 6 da `fact(3)` e lo assegna alla variabile fz

La Ricorsione

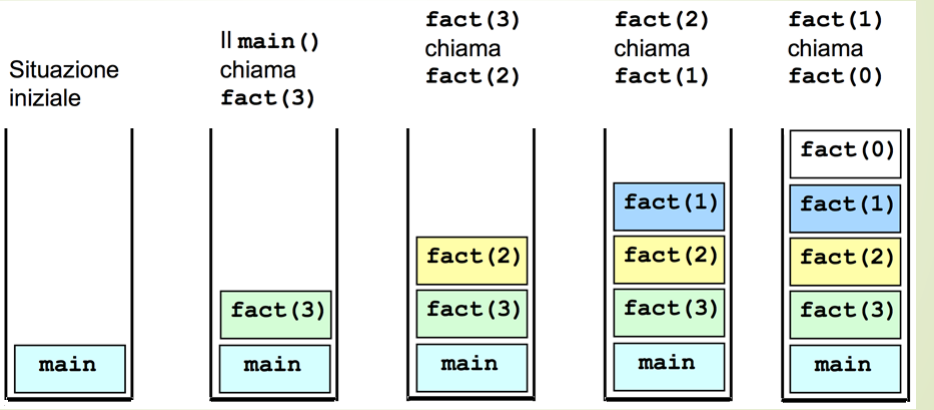
Il Fattoriale



main	fact(3)= 3 * fact(2)= 2 * fact(1) = 1 *fact(0)			
Cliente di fact(3)	Cliente di fact(2) Servitore del main	Cliente di fact(1) Servitore di fact(3)	Cliente di fact(0) Servitore di fact(2)	Servitore di fact(1)

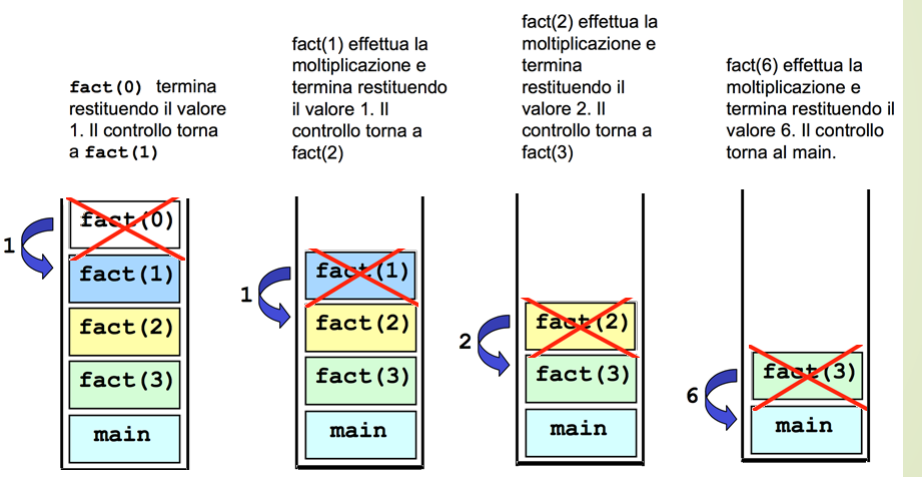
La Ricorsione

Il Fattoriale – Cosa succede nello stack?



La Ricorsione

Il Fattoriale – Cosa succede nello stack?



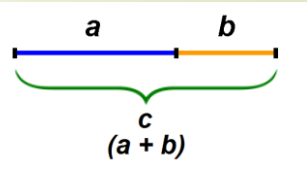
Funzioni Ricorsive
Fibonacci

- La successione di Fibonacci (aurea) è una successione di interi positivi in cui ciascun numero è la somma dei due precedenti
  - i primi due sono (per definizione) 1
- Definizione ricorsiva:
  - $F_0 = 1$
  - $F_1 = 1$
  - $F_n = F_{n-1} + F_{n-2}$ per ogni $n > 1$
- I primi termini della successione sono: 1,1,2,3,5,8,13,21,34,55,89,144

Funzioni Ricorsive
Fibonacci

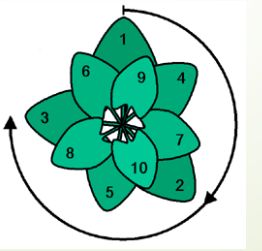
- Leonardo Fibonacci (1175 – 1235) era un matematico pisano
- Introdusse la serie per trovare una legge matematica che descrivesse la crescita di una popolazione di conigli
- Curiosità: il rapporto fra le coppie di termini successivi tende molto rapidamente al numero 1,61803..., noto con il nome di rapporto aureo o sezione aurea.

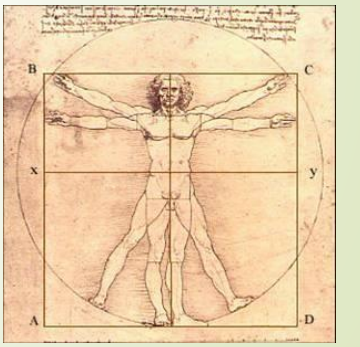
Sezione aurea



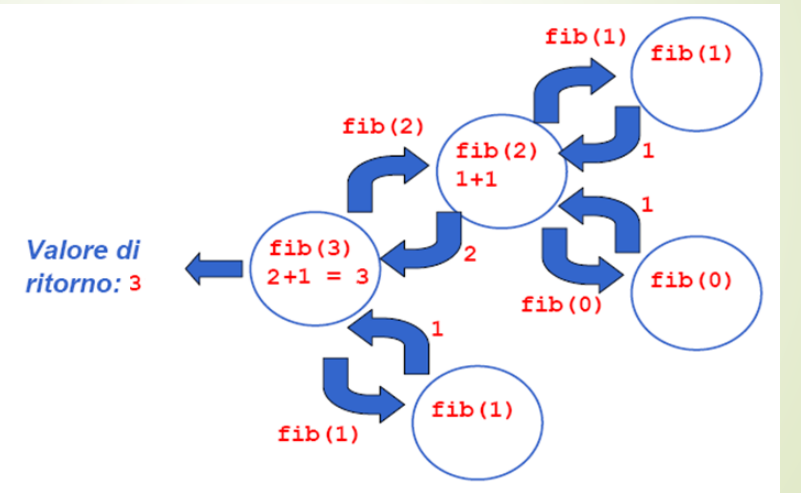
- La *sezione aurea* è ottenuta effettuando il rapporto fra due lunghezze disuguali delle quali la maggiore a è medio proporzionale tra la minore b e la somma delle due (a+b)

$$\frac{a+b}{a} = \frac{a}{b} \stackrel{\text{def}}{=} \varphi$$





La Ricorsione
Fibonacci



Ricorsione
Riflessioni

- Negli esempi visti finora, si inizia a sintetizzare il risultato «a ritroso», solo dopo che le chiamate si sono chiuse
- Il risultato viene sintetizzato a partire dalla fine, perché occorre prima arrivare al caso «banale»
  - Il caso banale fornisce il valore di partenza
  - Poi si sintetizzano a ritroso i successivi risultati parziali

Esercizi

- Algoritmo di Euclide
- Stringa palindroma

Algoritmo di Euclide

- Per trovare il Massimo Comune Divisore (MCD)
- Dati due numeri naturali a e b
  - Se b=0, a è il MCD
  - Altrimenti, si divide a / b e si assegna ad r il resto della divisione.
    - Se r = 0, b è il MCD
    - Altrimenti, occorre assegnare a = b e b = r e si ripete nuovamente la divisione.
- Sviluppare un programma che calcoli il MCD di due numeri dati da riga di comando. Scrivere sia la versione iterativa sia quella ricorsiva per la funzione MCD

Algoritmo di Euclide
Esempio

- Calcolare in MCD tra
  - a = 494
  - b = 130
- MCD (a, b) = 26

a	b	r
494	130	104
130	104	26
104	26	0
26	0	

Stringa Palindroma

- Una **stringa palindroma** è una sequenza di caratteri che, letta al contrario, rimane invariata.
  - Es: aerea, anilina, onorarono, radar, sos.
  - Es. frase: i topi non avevano nipoti.
- Scrivere le versioni iterativa e ricorsiva di una funzione che determini se una stringa in ingresso è palindroma o no, ovvero se il primo carattere e' uguale all'ultimo, il secondo carattere e' uguale al penultimo e cosi' via.

Agenda

- Record di attivazione
- Ricorsione
- Iterazione VS Ricorsione
- Ricorsione Tail
- Strutture Ricorsive

Iterazione: Fattoriale (cont.)

Costruiamo ora una funzione che calcola il fattoriale in modo iterativo

```
int fact(int n){
      int i=1;
      int F=1; /*inizializzazione del fattoriale*/
      while (i <= n)
            { F=F*i;
              i=i+1; }
      return F;
      }
```

DIFFERENZA CON LA VERSIONE RICORSIVA: ad ogni passo viene accumulato un risultato intermedio

La variabile F accumula risultati intermedi: se n = 3 inizialmente `F=1`, *poi al primo ciclo* `F=1`, *poi al secondo ciclo* `F` *assume il valore 2. I ne all'ultimo ciclo* `i=3` *e* `F` *assume il valore 6*

- `Al primo passo F accumula il fattoriale di 1`
- `Al secondo passo F accumula il fattoriale di 2`
- `Al passo i-esimo F accumula il fattoriale di i`

21

Iterazione

- Nell'esempio precedente il risultato viene sintetizzato "*in avanti*"
- L'esecuzione di un algoritmo di calcolo che computi "in avanti", per accumulo, è un *processo computazionale iterativo*.
- La caratteristica fondamentale di un *processo computazionale iterativo* è che *a ogni passo è disponibile un risultato parziale*
  - dopo k passi, si ha a disposizione il risultato parziale relativo al caso k
  - questo *non è vero* nei processi computazionali *ricorsivi*, in cui nulla è disponibile finché non si è giunti fino al caso elementare.

Iterazione (cont.)

- Un processo computazionale iterativo si può realizzare anche tramite funzioni ricorsive.
- Si basa sulla disponibilità di una variabile, detta accumulatore, destinata ad esprimere in ogni istante la soluzione corrente.
  - L'accumulatore viene passato come parametro ad ogni chiamata della funzione.

Agenda

- Record di attivazione
- Ricorsione
- Iterazione VS Ricorsione
- Ricorsione Tail
- Strutture Ricorsive

35

Ricorsione Tail

- Una ricorsione che realizza un processo computazionale iterativo è una *ricorsione apparente*.
- La chiamata ricorsiva è sempre l'ultima istruzione
  - I calcoli sono fatti prima
  - La chiamata serve solo, dopo averli fatti, per proseguire la computazione
- Questa forma di ricorsione si chiama «*Ricorsione Tail*» (ricorsione in coda)

36

Ricorsione Tail

- **il corpo del ciclo rimane *immutato***
- **il ciclo diventa un `if` con, in fondo, la chiamata tail-ricorsiva**

`while (condizione) {`
`  <corpo del ciclo>`
`}`
	`if (condizione) {`
	`  <corpo del ciclo>`
	`  <chiamata ricorsiva>`
	`}`

Naturalmente, può essere necessario ***aggiungere nuovi parametri*** nell'intestazione della funzione tail-ricorsiva, per "portare avanti" le variabili di stato

Ricorsione Tail: Il Fattoriale (cont.)

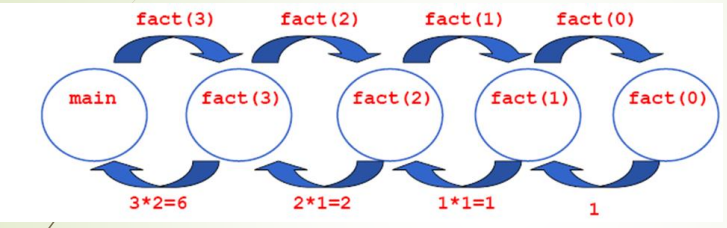
```
int fact(int n){
  return factIt(n,1,1);
}

int factIt(int n, int F, int i){
  if (i <= n)
   {F = i*F;
    i = i+1;
    return factIt(n,F,i);
   }
  return F;
}
```

Inizializzazione dell'accumulatore: corrisponde al fattoriale di 1

Contatore del passo

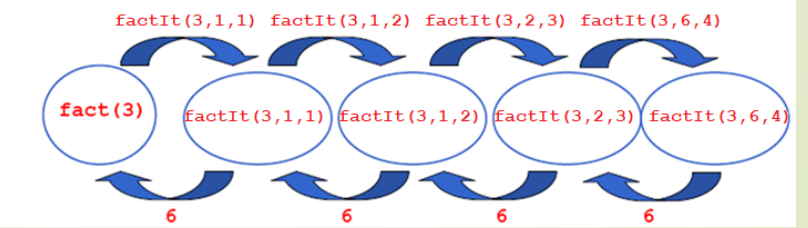
Accumulatore del risultato parziale

Ricorsione Tail: Il Fattoriale (cont.)



Ricorsione classica

Ricorsione Tail



Ricorsione Tail: Il Fattoriale (cont.)

- La soluzione ricorsiva individuata per il fattoriale è sintatticamente ricorsiva ma dà luogo ad un processo computazionale iterativo.
  - Ricorsione apparente, detta «Ricorsione Tail»
- Il risultato viene sintetizzato «in avanti»
  - Ogni passo decompone e calcola
  - E porta in avanti il nuovo risultato parziale
  - Quando le chiamate si chiudono non si fa altro che riportare indietro, fino al chiamante, il risultato ottenuto

Agenda

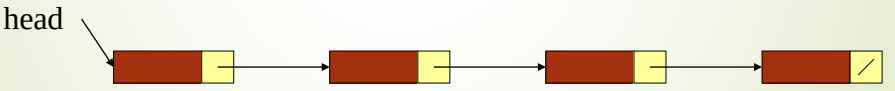
- Record di attivazione
- Ricorsione
- Iterazione VS Ricorsione
- Ricorsione Tail
- Strutture Ricorsive

Strutture Ricorsive

- Alcune strutture dati sono inerentemente ricorsive:
  - Sequenze
  - Strutture ad albero
  - …
- La formulazione ricorsiva di algoritmi su di esse risulta più naturale
- Esempio: definizione di Lista. Una lista è…
  - Una lista vuota, oppure
  - Un elemento seguito da una lista.

Strutture Ricorsive
Lista linkata

- La struttura in basso è una lista con puntatore al primo elemento head
- È anche un elemento (quello puntato da head) seguito da una lista (puntata da head->next).



Algoritmi ricorsivi su lista

- **Stampa lista**: stampo l'elemento corrente p e chiamo ricorsivamente la funzione di stampa sulla lista puntata da p->next
- **Ricerca**: verifico se il dato cercato è presente nell'elemento corrente (in caso affermativo restituisco l'elemento) e chiamo ricorsivamente la funzione di ricerca sulla lista puntata da p->next
- **Numero di occorrenze di un item**: verifico se il dato cercato è presente nell'elemento corrente (in caso affermativo incremento il contatore) e chiamo ricorsivamente la funzione di conteggio sulla lista puntata da p->next;
- **Deallocazione degli elementi**: chiamo ricorsivamente la funzione di deallocazione sulla lista puntata da p->next e libero la memoria corrispondente all'elemento corrente p

Ricorsione vs. Iterazione

- Ripetizione
  - Iterazione: ciclo esplicito
  - Ricorsione: chiamate di funzione ripetute
- Terminazione
  - Iterazione : il ciclo fallisce la condizione
  - Ricorsione : il caso base è riconosciuto
- Entrambe possono dar luogo a cicli infiniti
- Bilancio: Scegli tra performance (iterazione) e buona ingegneria del software (ricorsione)
  - La ricorsione richiede un notevole sovraccarico (overhead) al tempo di esecuzione dovuto alle chiamate a funzione.

Ricorsione vs. Iterazione

- Algoritmi che per loro natura sono ricorsivi piuttosto che iterativi dovrebbero essere formulati con procedure ricorsive
- Ad esempio, alcune strutture dati sono inerentemente ricorsive:
  - Strutture ad albero
  - Sequenze
  - …
- La formulazione ricorsiva di algoritmi su di esse risulta più naturale
- La ricorsione deve essere evitata quando esiste una soluzione iterativa ovvia e in situazioni in cui le prestazioni del sistema sono un elemento critico

Esercizio

- **Scrivere una funzione ricorsiva print_rev che, data una stringa, ne stampi i caratteri in ordine inverso.**
- **Ad esempio:**