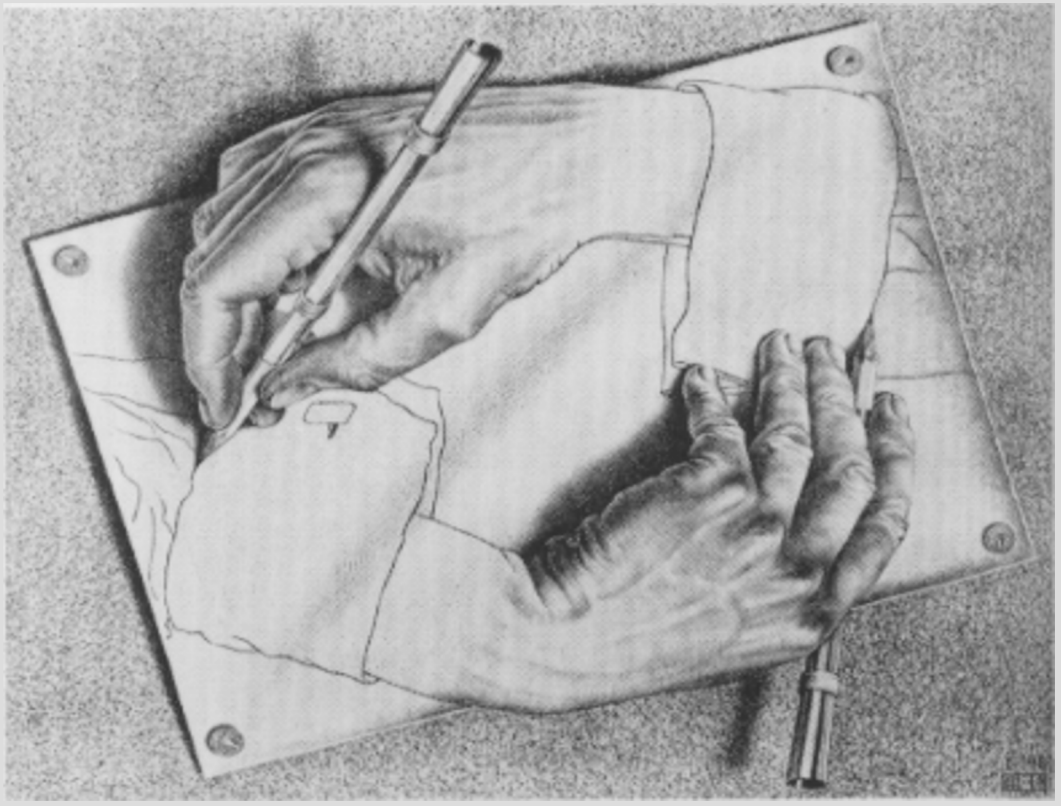
Parte 2

Ricorsione



[M.C.Escher – Drawing hands, 1948]

Funzioni ricorsive

- Una funzione si dice ***ricorsiva*** se richiama se stessa, direttamente o indirettamente
- La ricorsione si dice diretta se la funzione invoca se stessa senza intermediari
- La ricorsione si dice indiretta se coinvolge più di una funzione

ESEMPIO

Funzione uno() che invoca una funzione due() che a sua volta richiama uno()

Esempio

```
void fun(int n)
{  cout<<n<<endl ;
    if (n>0)
    fun(n-1) ;
}

int main ()
{   fun(2) ;
    return 0 ;
}
```

Funzione ricorsiva (diretta)

Che cosa fa il programma?

Vedi programma *ric1.cc*

Usa il debbuger per vedere la sequenza delle chiamate

Soluzioni ricorsive

La ricorsione è una ***soluzione concisa ed elegante*** per alcune tipologie di problemi (***non adatta a tutte!***)

- Adatta a problemi complessi scomponibili in sottoproblemi più semplici
- Utile per svolgere compiti ripetitivi su di un set di input variabili

Alcuni problemi hanno una ***natura intrinsecamente ricorsiva***

Ad esempio il calcolo del fattoriale di un numero intero, la funzione di Fibonacci, la Torre di Hanoi, ….

Calcolo del fattoriale

n! = n* (n-1) * (n-2) * … * 1

5! = 5 * 4 * 3 * 2 * 1 = 5 * 4! = 120

(5 – 1)!

n! = n * (n-1)!

Intrinseca natura ricorsiva: si calcola il fattoriale di n calcolando il fattoriale di n-1

Pensate a come scrivere la funzione...

Funzione ricorsiva fattoriale

```
/* Legge in ingresso un numero intero n non
negativo, e ritorna il valore n!  */

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

Cosa succederebbe senza queste istruzioni?

Vedi programma *fatt_ricorsivo.cc*

Catena infinita di chiamate ricorsive!
Manca la *condizione di terminazione*

Condizione di terminazione

- Elemento *necessario* in una funzione ricorsiva
- Determina la *fine della catena* delle chiamate ricorsive
- Verifica se siamo in presenza di un **caso base** che viene eseguito quando diventa vera la condizione di terminazione
- Può identificare semplicemente la fine della catena ricorsiva
- Può eseguire qualche specifica operazione

ESEMPIO

- Qual è la condizione di terminazione di `fun()`?

```
void fun(int n)
{   cout<<n<<endl ;
     if (n>0)
     fun(n-1) ;
}
```

Argomento di controllo

- La ricorsione è controllata da uno o più parametri della funzione, detti **argomenti di controllo**

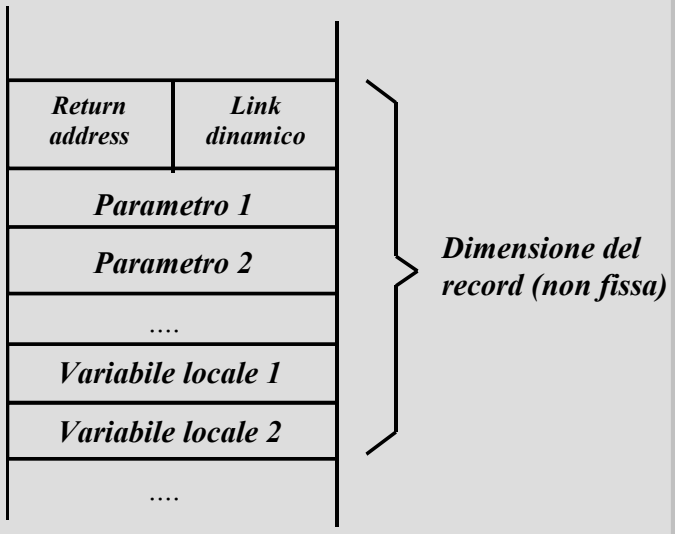
```
int fattoriale (int n)
```

- In funzione del valore dell'argomento di controllo si determina se eseguire un caso base o la chiamata ricorsiva
- Nel caso della chiamata ricorsiva, la funzione viene invocata passandole un *nuovo valore dell'argomento di controllo*
- La *sequenza* dei valori dell'argomento di controllo di chiamate successive deve tendere verso uno dei casi basi altrimenti la funzione diverge

```
fattoriale(3)-> fattoriale(2)->
                fattoriale(1)-> fattoriale(0)
```

Recap: Record di attivazione

- Quando una funzione viene invocata, alcune informazioni vengono salvate sullo **stack**:
  - il punto del codice in cui è stata invocata (indirizzo di ritorno o **Return address - RA**)
  - i parametri e le variabili locali
- L'insieme di questi dati sullo stack è detto **Record di Attivazione RdA**

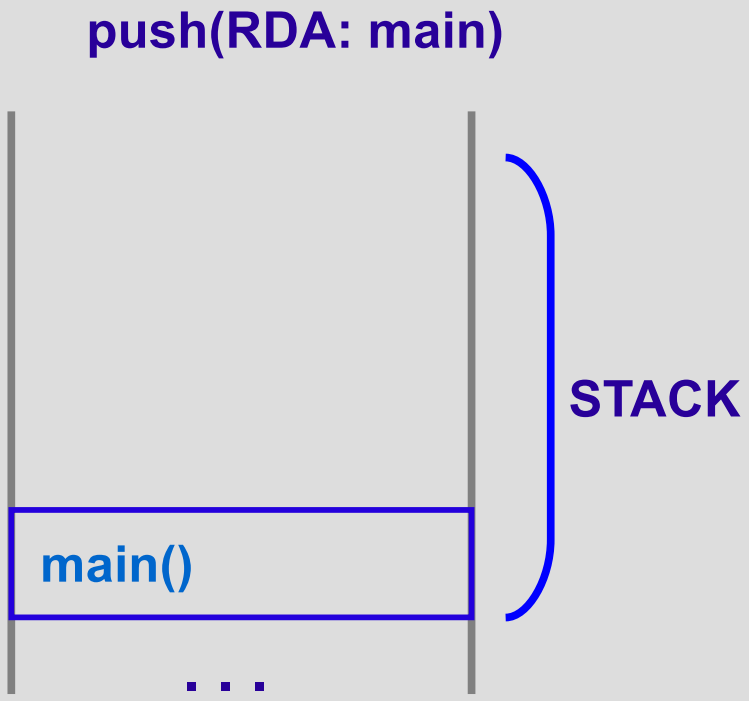


- Cosa succede nel caso della ricorsione?
- Possono esistere più istanze della stessa funzione nello stesso istante (*istanze multiple*)

Esempio: fun()

```
void fun(int n)
{ cout<<n<<endl ;
  if (n>0)
     fun(n-1) ;
}

main () {
     fun(2) ;
}
```



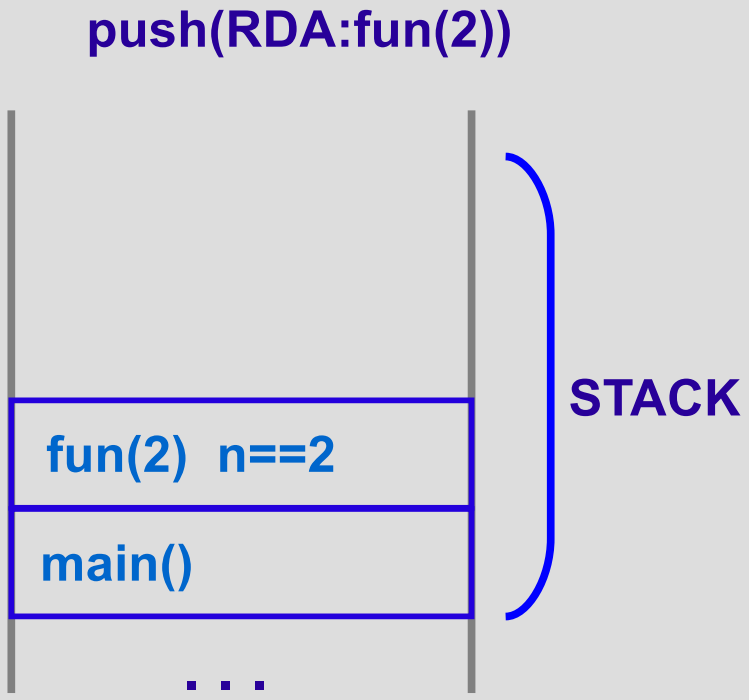
Sequenza di attivazioni:

(S.O.)->main()->fun(2)->fun(1)->fun(0)

Esempio: fun()

```
void fun(int n)
{ cout<<n<<endl ;
  if (n>0)
     fun(n-1) ;
}

main () {
     fun(2) ;
}
```



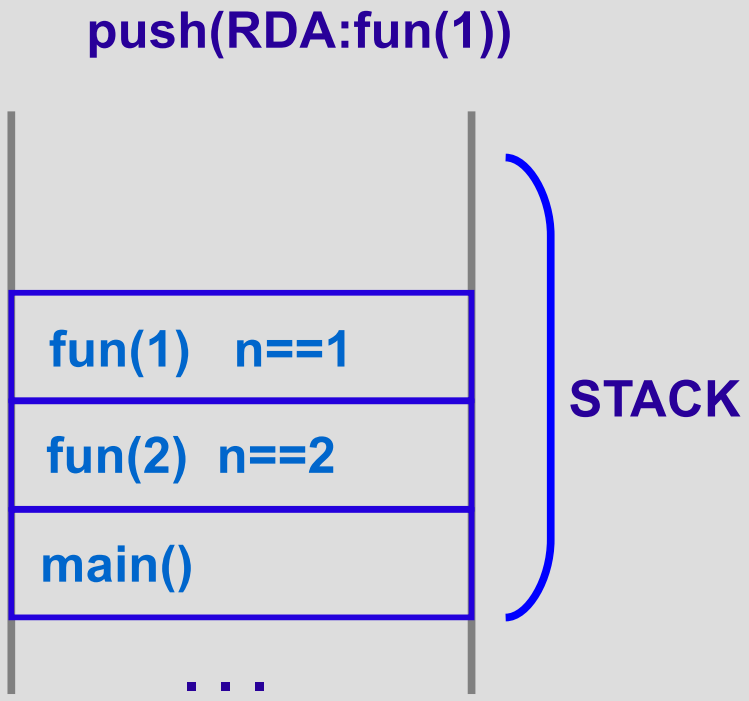
Sequenza di attivazioni:

(S.O.)->main()->fun(2)->fun(1)->fun(0)

Esempio: fun()

```
void fun(int n)
{ cout<<n<<endl ;
  if (n>0)
     fun(n-1) ;
}

main () {
     fun(2) ;
}
```



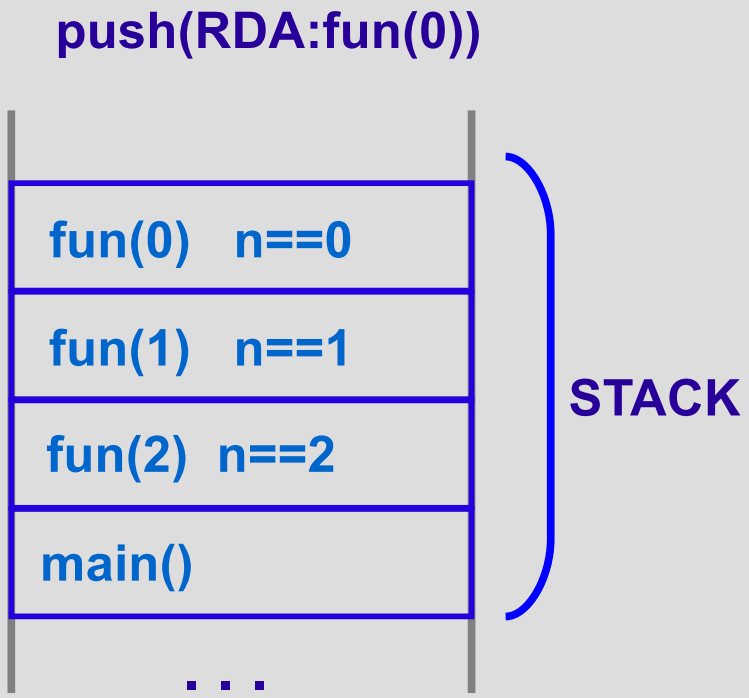
Sequenza di attivazioni:

(S.O.)->main()->fun(2)->fun(1)->fun(0)

Esempio: fun()

```
void fun(int n)
{ cout<<n<<endl ;
  if (n>0)
     fun(n-1) ;
}

main () {
     fun(2) ;
}
```



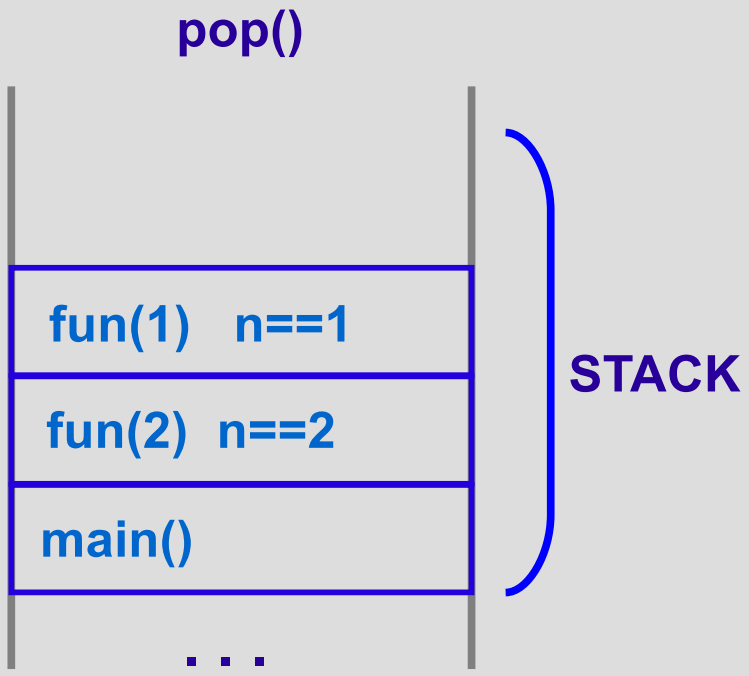
Sequenza di attivazioni:

(S.O.)->main()->fun(2)->fun(1)->fun(0)

Esempio: fun()

```
void fun(int n)
{ cout<<n<<endl ;
  if (n>0)
     fun(n-1) ;
}

main () {
     fun(2) ;
}
```



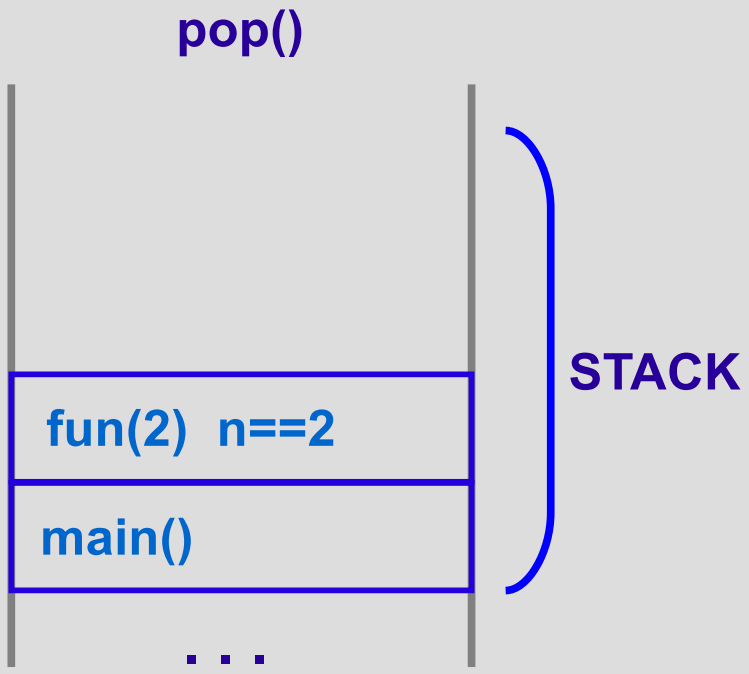
Sequenza di attivazioni:

(S.O.)->main()->fun(2)->fun(1)->fun(0)

Esempio: fun()

```
void fun(int n)
{ cout<<n<<endl ;
  if (n>0)
     fun(n-1) ;
}

main () {
     fun(2) ;
}
```



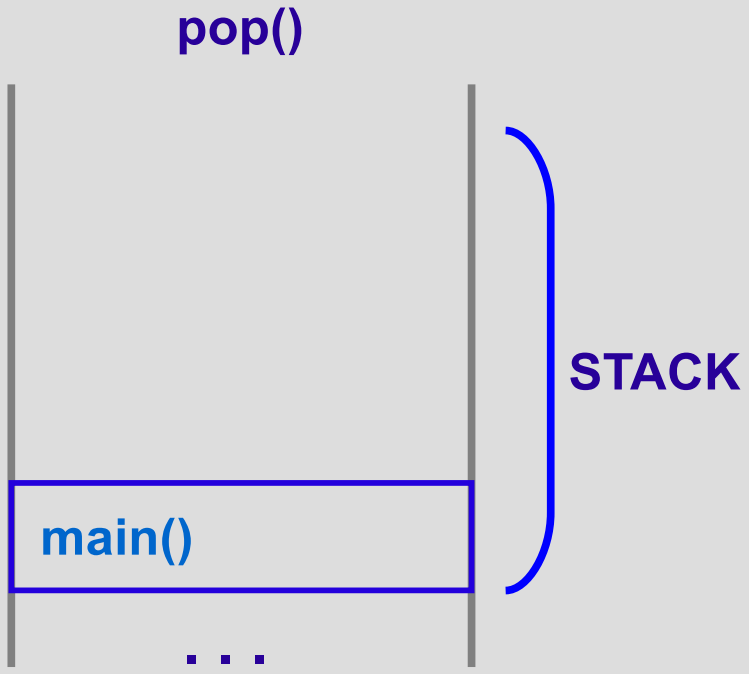
Sequenza di attivazioni:

(S.O.)->main()->fun(2)->fun(1)->fun(0)

Esempio: fun()

```
void fun(int n)
{ cout<<n<<endl ;
  if (n>0)
     fun(n-1) ;
}

main () {
     fun(2) ;
}
```



Sequenza di attivazioni:

(S.O.)->main()->fun(2)->fun(1)->fun(0)

Esercizi

Provare ad eseguire gli esercizi sulle funzioni ricorsive presenti nelle prove scritte degli appelli passati.

I NUMERI DI FIBONACCI

Nel 1223 a Pisa, Leonardo Pisano, meglio noto come **Fibonacci**, vinse un singolare torneo tra abachisti e algoritmisti, armati soltanto di carta, penna e pallottoliere.

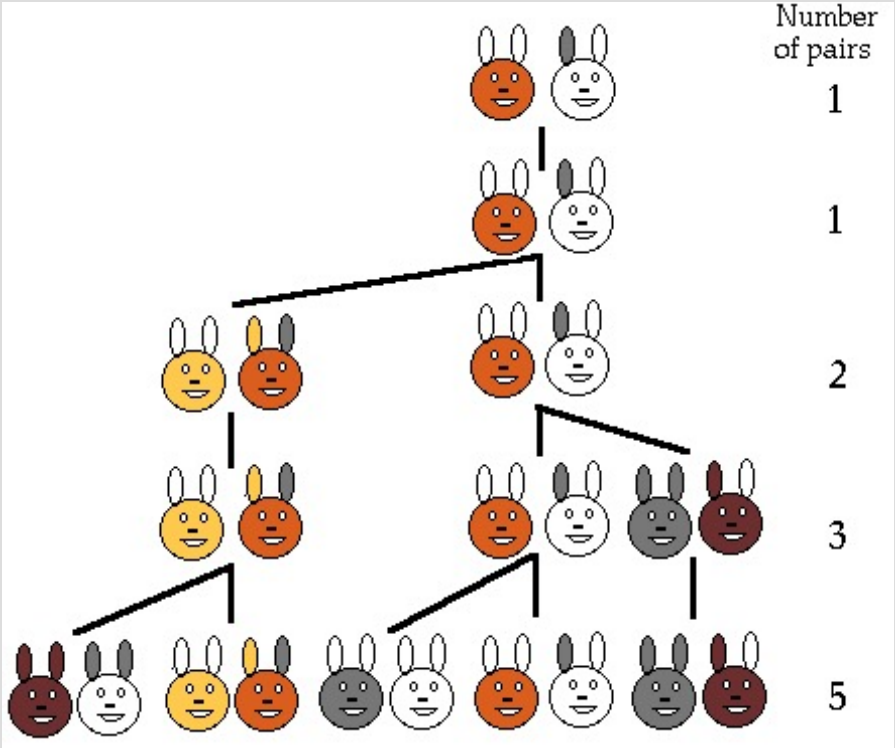
In quella gara si dimostrò che col metodo posizionale indiano appreso dagli arabi si poteva calcolare più velocemente di qualsiasi abaco.

Il quesito era il seguente:

“Quante coppie di conigli si ottengono in un anno a partire da una singola coppia supponendo che

- le coppie non muoiano mai
- ogni coppia dia alla luce un'altra coppia ogni mese
- le coppie più giovani siano in grado di riprodursi dal secondo mese di vita?"

I NUMERI DI FIBONACCI

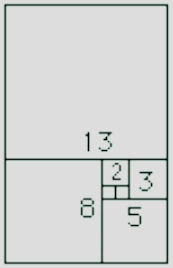


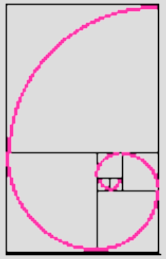
$$fib(n) = \begin{cases} n & se\ n = 0\ o\ n = 1 \\ fib(n-1) + fib(n-2) & altrimenti \end{cases}$$

La serie di Fibonacci

- Ha riscontro in vari fenomeni naturali.
  Ad esempio il numero dei petali di molte piante (3 per iris e lilium, 21 per cicoria, ecc.) o il numero dei semi delle piante (girasoli, ecc.).

- Descrive una forma di spirale





- Il rapporto tra i numeri consecutivi di Fibonacci converge verso
  la sezione aurea *o* ***costante di Fidia***

$$\lim_{n\to\infty} \frac{F_n}{F_{n-1}} = \phi$$

$$\phi = \frac{1+\sqrt{5}}{2} = 1,6180339887...$$

La sezione aurea:

- corrisponde al rapporto tra la lunghezza del braccio e l'avambraccio
- … e altre proporzioni sono verificabili per il corpo umano
- è usata in architettura per progettare finestre, stanze, palazzi, ecc.

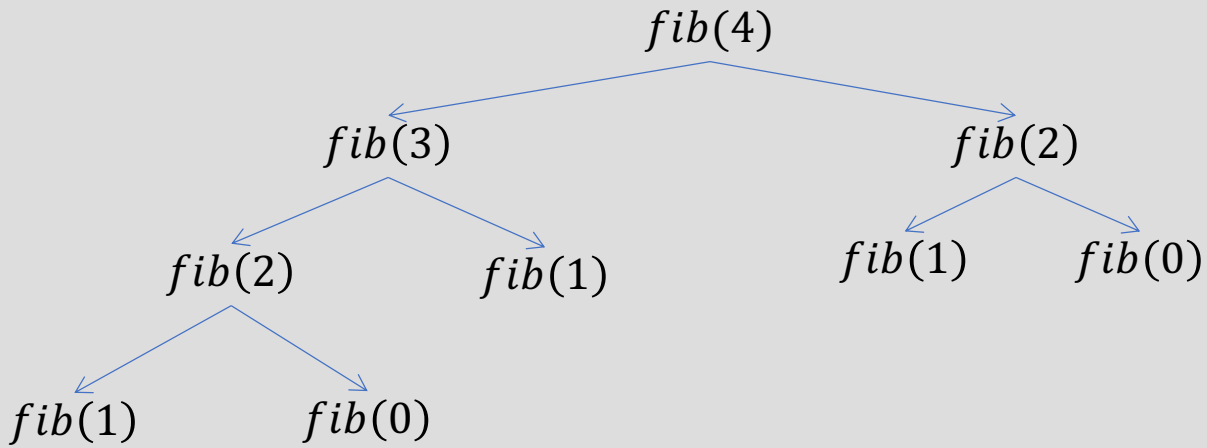
Esercizio

Scrivere un programma che chiede all'utente un numero e stampa il risultato della funzione di Fibonacci per quel numero

SOLUZIONE **Vedi programma** *`fibonacci.cc`*

Ricorsione multipla

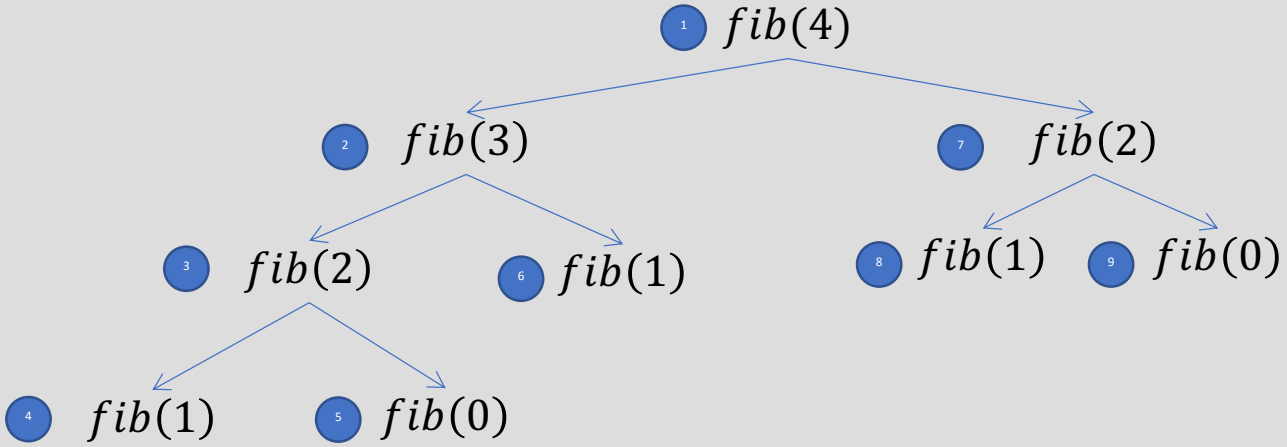
- La funzione di Fibonacci è un classico esempio di ricorsione multipla



Qual è la sequenza di RdA sullo stack?
SUGGERIMENTO: Usa il debugger

Ricorsione multipla

- La funzione di Fibonacci è un classico esempio di ricorsione multipla




```
push(fib(4))
push(fib(3))
push(fib(2))
push(fib(1))
pop()
push(fib(0))
pop()
pop()
push(fib(1))
pop()
pop()
push(fib(2))
push(fib(1))
pop()
push(fib(0))
pop()
pop()
pop()
```

Qual è la sequenza di RdA sullo stack e quali sono i valori restituiti dalle singole chiamate?

SUGGERIMENTO: Usa il debugger

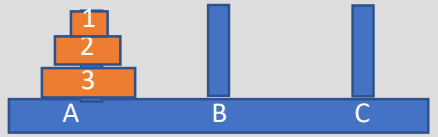
La torre di Hanoi

- La Torre di Hanoi è un *rompicapo matematico* composto da tre paletti e un certo numero di dischi di grandezza decrescente, che possono essere infilati in uno qualsiasi dei paletti.
- Il gioco inizia con tutti i dischi incolonnati su un paletto in ordine decrescente, in modo da formare un cono.
- Lo scopo del gioco è portare tutti i dischi sull'ultimo paletto.
- E’ possibile spostare solo un disco alla volta e potendo mettere un disco solo su un altro disco più grande, mai su uno più piccolo

La soluzione del gioco

La soluzione base del gioco della torre di Hanoi si formula in modo ricorsivo: **ALGORITIMO PER LA TORRE DI HANNOI**

Siano i paletti etichettati con A, B e C, e
i dischi numerati da 1 (il più piccolo) a n (il più grande)

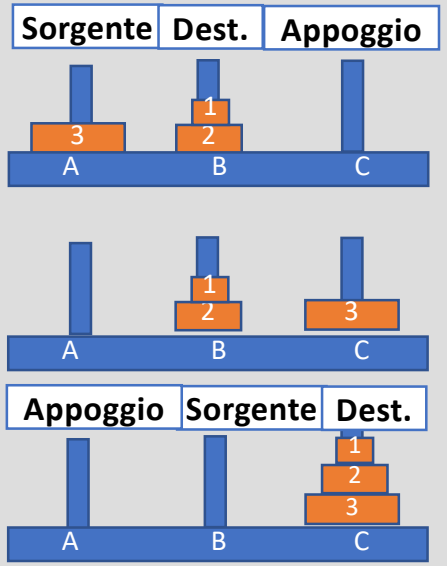


Per spostare n dischi da A a C usando B come appoggio

① Sposta i primi n-1 dischi da A a B. (Questo lascia il disco n da solo sul paletto A) usando come appoggio il paletto C

② Sposta il disco n da A a C

③ Sposta n-1 dischi da B a C usando come appoggio il paletto A



La **soluzione del problema** per n dischi è stata ricondotta alla **soluzione del problema** per n-1 dischi (cfr. passi 1 e 3)!

Esercizio

Scrivere un programma che dato un numero di dischi stampa la sequenza di mosse per risolvere il gioco della torre di Hanoi

Quali sono i parametri della funzione?

`n`: numero di dischi sul piolo sorgente

`pioloS`: piolo sorgente

`pioloD`: piolo destinazione

`pioloA`: piolo appoggio

```
void hanoi(int n, char pioloS, char pioloD,
                  char pioloA)
```

Esercizio (cont.)

Qual è il caso base?

Il caso base è quello in cui ho un solo disco da muovere (n=1) e in questo caso sposto il disco da pioloS a pioloD

Quali sono le due chiamate ricorsive?

SOLUZIONE Vedi programma *hanoi.cc*

Iterazione vs. ricorsione

Entrambe implicano la ripetizione e si basano su un test di controllo

Ricorsione → concisa ed elegante per problemi scomponibili

Iterazione → meno concisa per lo stesso tipo di problemi, ma più efficiente nello spazio e nel tempo...
Perchè?

Non è necessario allocare spazio in memoria
per un nuovo record di attivazione
ad ogni chiamata ricorsiva

Esercizio

Trasformiamo alcune funzioni e primitive sulle liste implementate in maniera iterativa con soluzioni ricorsive.

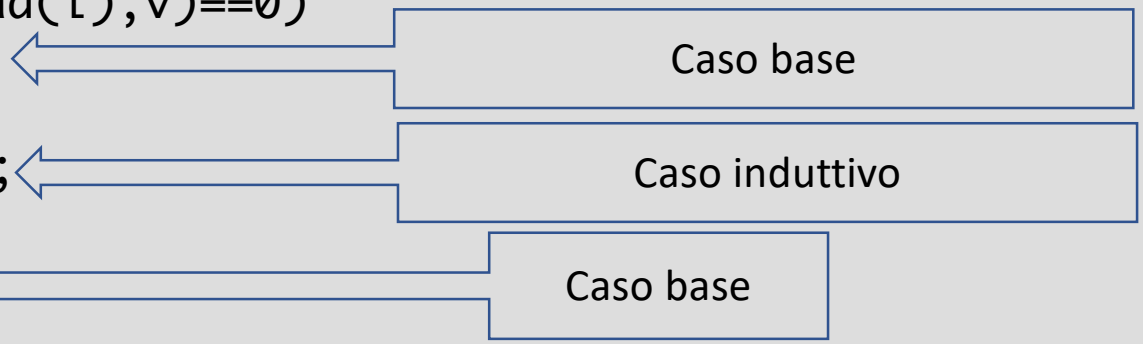
- Per le liste semplici `delete_elem()` `merge()` `search()`
- Per le liste doppie `search()`

Esercizio (cont.)

Vediamo il caso di `search()`:

```
elem* search(lista l, char* v){
    while(l!=NULL)
        if(strcmp(head(l),v)==0)
            return l;
        else
            l=tail(l);

    return NULL;}
```



ESERCIZIO Modificare il programma *gestione_lista_doppia_sol.cc*

sostituendo la funzione `search()` dalla versione iterattiva alla versione ricorsiva

Esercizio (cont.)

```
elem* search(lista l, char* v){

        if(l == NULL)
                return NULL;
        if(strcmp(head(l),v)==0)
            return l;
         return search(tail(l),v);
    }
```

SOLUZIONE Vedi programma

gestione_lista_doppia_ricorsiva.cc

Ricorsione: considerazioni

Buone norme di programmazione:

- Verificare la correttezza della condizione di terminazione per evitare ricorsioni infinite
- Tenere sotto controllo lo stack
- Usare la ricorsione solo per problemi adatti
- Limitare la ricorsione ad una funzione
- Ricorsioni mutue sono pericolose in quanto difficili da gestire e/o rilevare

Esercizio

Scrivere una funzione ricorsiva che effettua il calcolo della potenza di un numero reale elevato ad esponente intero, positivo o nullo.

Potenza ricorsiva: suggerimenti

Si può scrivere a^3 in funzione di a^2?

a^3=a*a^2

In generale:

a^n=a*a^(n-1)

Qual è la condizione di terminazione?

SOLUZIONE **Vedi programma** *potenza_ric.cc*

Programmi

Stampa tutti i numeri da n a 1, con n inserito da utente

Vedi programma *Stampa_ric_decresc.cc*

Stampa tutti i numeri da 1 ad n, con n inserito da utente

Vedi programma *Stampa_ric_cresc.cc*

Esercizi non risolti

- Scrivere la funzione ricorsiva `int lunghezza(lista l)` che, data una lista, restituisce la sua lunghezza (attenzione: una lista può anche essere vuota)
- Modificare tutti I programmi sulle liste visti a lezione riscrivendo le funzioni/primitive iterative con soluzioni ricorsive