Cenni di Ricorsione

Informatica B

Francesco Trovò

francesco1.trovo@polimi.it

26 Novembre 2022

Funzioni: Riepilogo

Esempio: Definizione

```
function f = fattoriale(n)
f = 1;
for ii = 2 : n
    f = f * ii;
end
```

Parametro formale in uscita

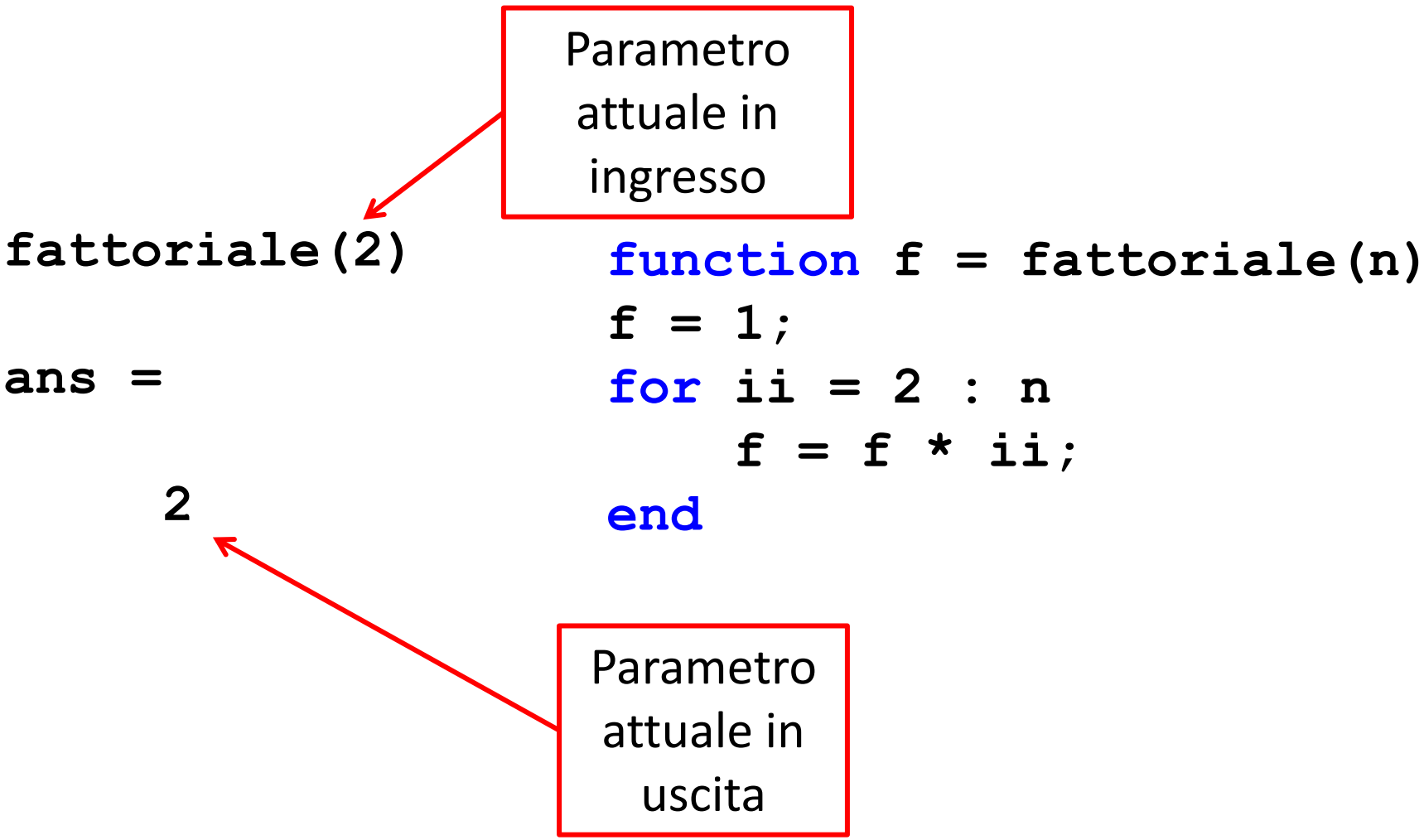
Parametro formale in ingresso

Definizione di $n!$

Il fattoriale di un intero $n > 0$ è definito come segue:

$$n! = n * (n-1) * (n-2) * \cdots * 2 * 1$$

Esempio: Invocazione



Un Esempio di Chiamata

```
k = 2;
f2 = fattoriale(k);
```

Workspace principale

- Da qui si invoca la funzione
- Contiene le variabili **`k`**, **`f2`**
- Non ha visibilità di **`f`**, **`ii`**, **`n`**

```
function f = fattoriale(n)
f = 1;
for ii = 2 : n
    f = f * ii;
end
```

Workspace locale

- Creato al momento dell'invocazione di fattoriale
- Contiene le variabili **`f`**, **`ii`**, **`n`**
- Non ha visibilità su **`k`**, **`f2`**
- Non ha legami con il workspace principale se non per i parametri attuali che vengono copiati (sia in ingresso che in uscita)
- Viene distrutto terminata l'esecuzione

Notiamo che...

- Vale la seguente relazione

$$n! = n * (n-1)!$$

si dimostra immediatamente dalla definizione di fattoriale

$$n! = n * \underbrace{(n-1) * (n-2) * \cdots * 2 * 1}_{(n-1)!}$$

- È possibile usare questa proprietà per definire un'implementazione di fattoriale totalmente diversa?

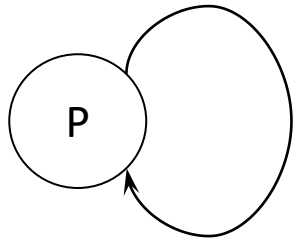
Ricorsione

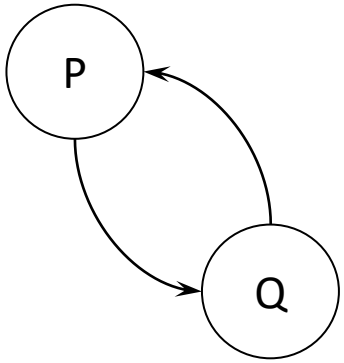
Esempio Reale di Ricorsione

https://en.wikipedia.org/wiki/Airplane!

Ricorsione

- Che cos'è la ricorsione?
  - un sottoprogramma P richiama se stesso (ricorsione diretta)
  - un sottoprogramma P richiama un sottoprogramma Q che comporta un'altra chiamata a P (ricorsione indiretta)





- È una tecnica di programmazione molto potente
- Permette di risolvere in maniera elegante problemi complessi
- Le funzioni che richiamano se stesse (direttamente o indirettamente) sono dette **funzioni ricorsive**

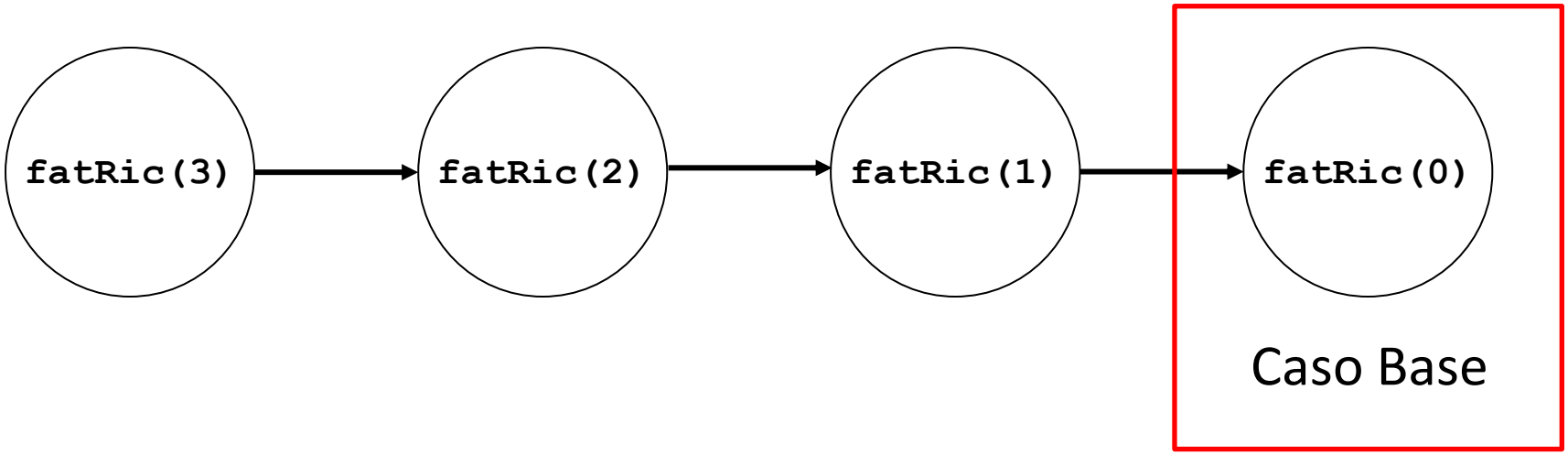
Esecuzione e Workspace Locali

- In ogni istante possono essere in corso **diverse attivazioni** dello **stesso** sottoprogramma
- Sono **tutte sospese tranne** una, **l'ultima invocata**, all'interno della quale si sta svolgendo il **flusso di esecuzione**
- Ogni attivazione esegue **lo stesso codice** ma opera su **workspace distinti**
- Si hanno quindi **copie distinte** dei **parametri attuali** e delle **variabili locali** nelle varie invocazioni

Loop Infinito

Domanda: se ogni volta la funzione richiama se stessa, perché la catena di invocazioni non continua **all'infinito**?

Risposta: la funzione ricorsiva deve prevedere una situazione in cui non richiama se stessa, i.e., il **caso base**



Programmazione Ricorsiva

Per risolvere un problema attraverso la programmazione ricorsiva sono **necessari alcuni elementi:**

- **Caso base**: caso elementare del problema che può essere risolto immediatamente
- **Passo ricorsivo**: chiamata ricorsiva per risolvere uno o più problemi più semplici
- **Costruzione della soluzione**: costruzione della soluzione sulla base del risultato delle chiamate ricorsive

Esempio: il Fattoriale

Definizione del fattoriale:

$$f(n) = n! = n * (n-1) * (n-2) * \cdots * 3 * 2 * 1$$

Definire caso base: di quale numero conosciamo il fattoriale senza calcolarlo?

$$f(0) = 1 \text{ (caso base)}$$

Definire passo ricorsivo: come facciamo a ricavare il fattoriale se conosciamo il fattoriale del numero precedente?

$$f(n) = n * f(n-1) \text{ (passo ricorsivo)}$$

Esempio: il Fattoriale Ricorsivo

```
function [f] = factRic(n)
if (n == 0)
  f = 1;
else
  f = n * factRic(n - 1);
end
```

Questa ci ricorda che $0! = 1$

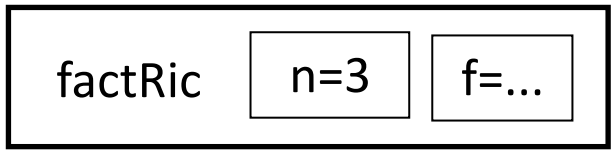
Questa ci ricorda che $n! = n * (n-1)!$

Usiamo una funzione che chiama se stessa (con un input differente)

Funzionamento Ricorsione (1)

```
function [f] = factRic(n)
  if (n == 0)
    f = 1;
  else
    f = n*factRic(n-1);
  end
```

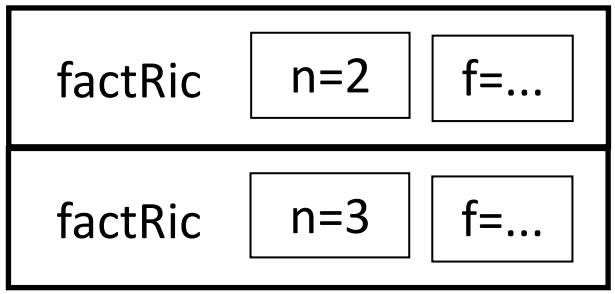
factRic(3)



Funzionamento Ricorsione (2)

```
function [f] = factRic(n)
  if (n == 0)
    f = 1;
  else
    f = n*factRic(n-1);
  end
```

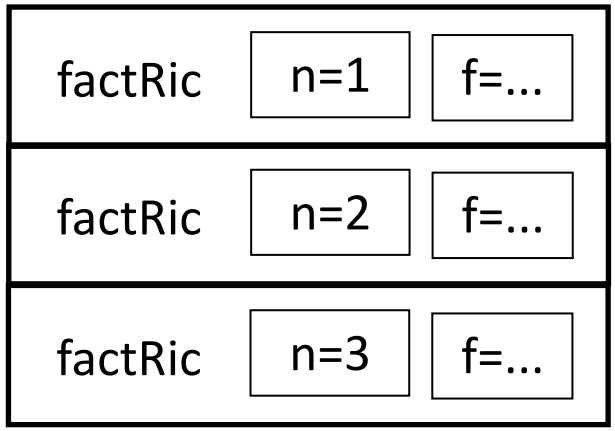
factRic(3)



Funzionamento Ricorsione (3)

```
function [f] = factRic(n)
  if (n == 0)
    f = 1;
  else
    f = n*factRic(n-1);
  end
```

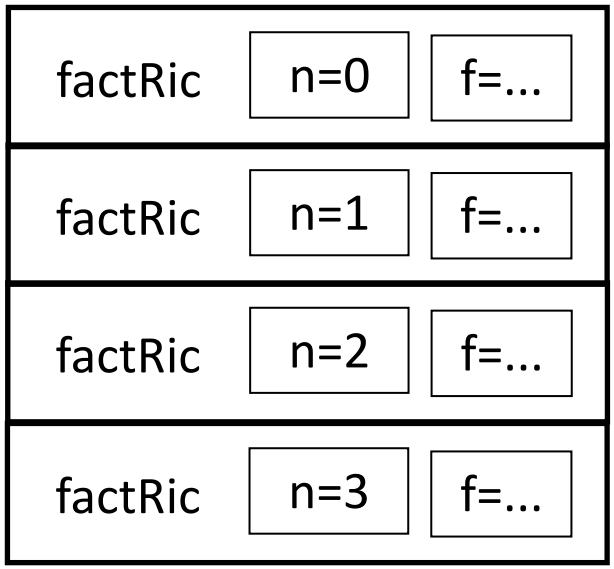
factRic(3)



Funzionamento Ricorsione (4)

```
function [f] = factRic(n)
  if (n == 0)
    f = 1;
  else
    f = n*factRic(n-1);
  end
```

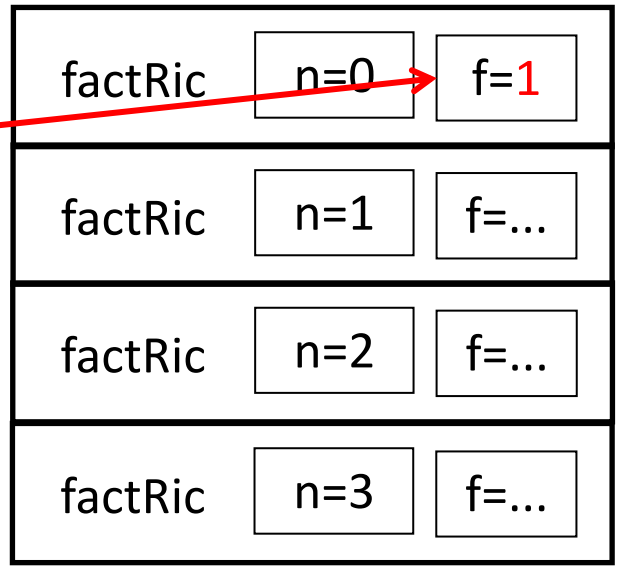
factRic(3)



Funzionamento Ricorsione (5)

```
function [f] = factRic(n)
  if (n == 0)
    f = 1;
  else
    f = n*factRic(n-1);
  end
```

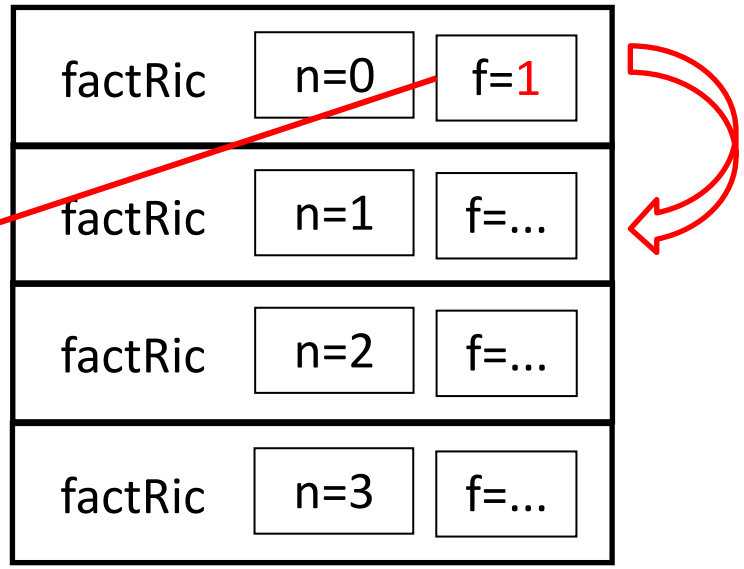
factRic(3)



Funzionamento Ricorsione (6)

```
function [f] = factRic(n)
  if (n == 0)
    f = 1;
  else
    f = n*factRic(n-1);
  end
```

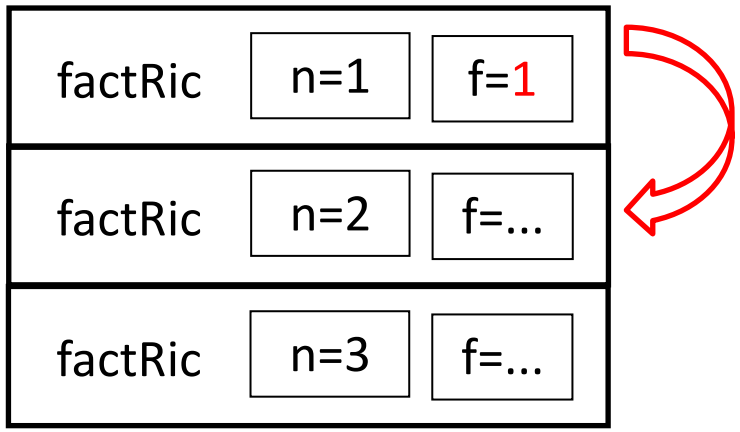
factRic(3)



Funzionamento Ricorsione (7)

```
function [f] = factRic(n)
  if (n == 0)
    f = 1;
  else
    f = n*factRic(n-1);
  end
```

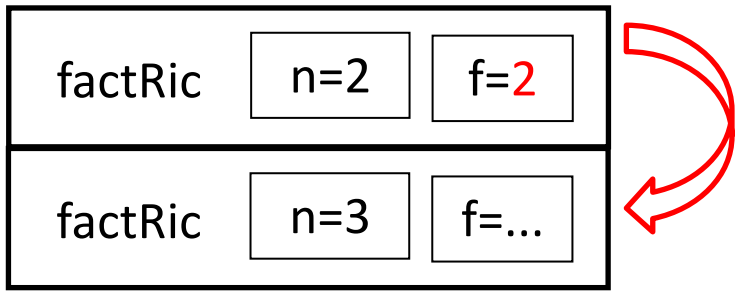
factRic(3)



Funzionamento Ricorsione (8)

```
function [f] = factRic(n)
  if (n == 0)
    f = 1;
  else
    f = n*factRic(n-1);
  end
```

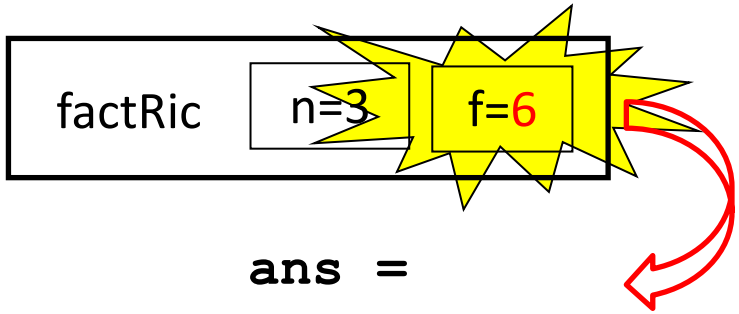
factRic(3)



Funzionamento Ricorsione (9)

```
function [f] = factRic(n)
  if (n == 0)
    f = 1;
  else
    f = n*factRic(n-1);
  end
```

```
factRic(3)
```




```
ans =

     6
```

Ricorsione in Coda

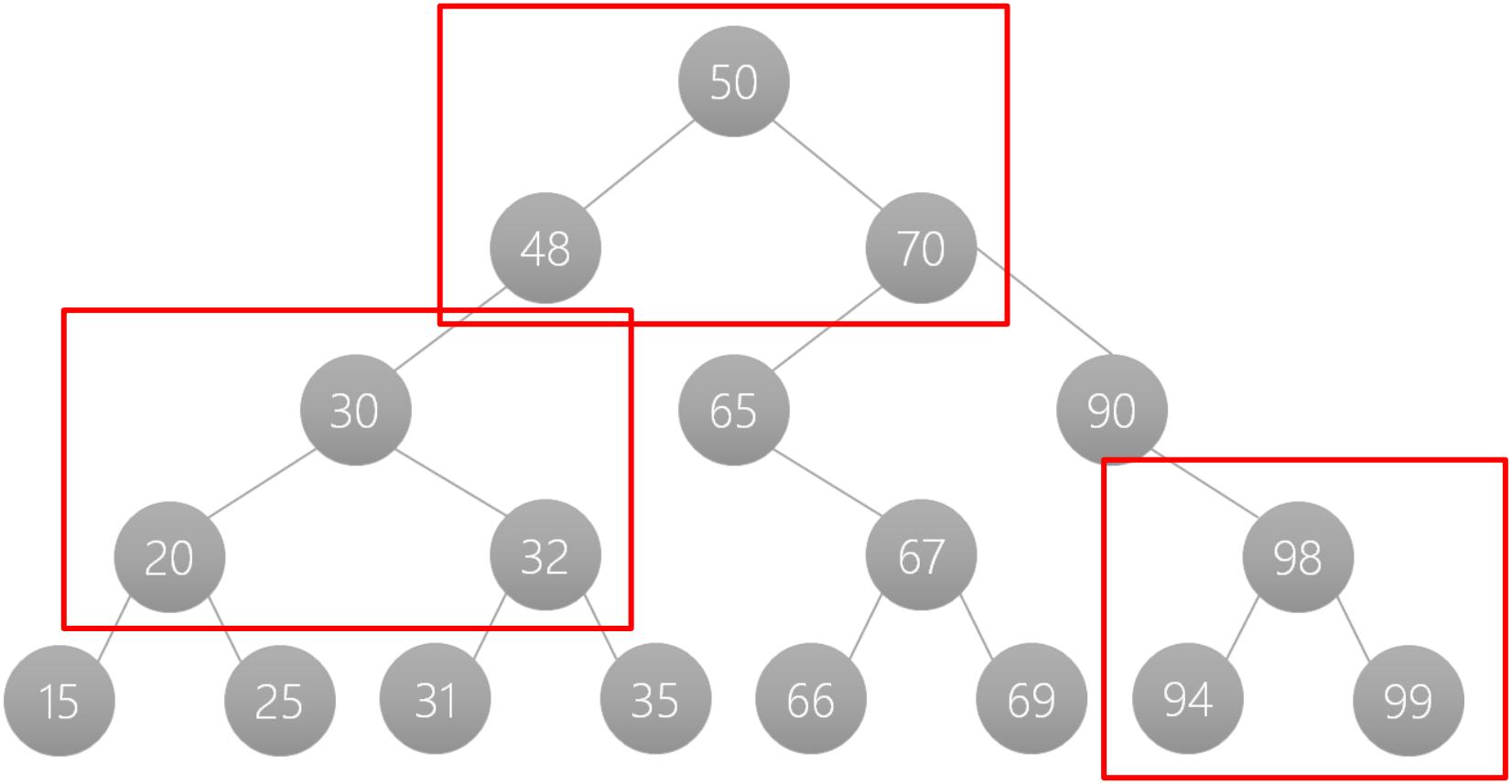
La ricorsione in coda:

- prevede **una sola chiamata ricorsiva**
- esegue la chiamata ricorsiva come **ultima istruzione**

```
function [f] = factRic(n)
  if (n == 0)
    f = 1;
  else
    f = n * factRic(n-1);
  end
```

A Cosa Serve la Ricorsione

- Ricerca su strutture ad albero: cerco se esiste il numero 15 nell'albero



Errori nella Ricorsione

Errori comuni nell'Uso della Ricorsione

Terminazione della catena ricorsiva:

- È presente il caso base?
- Viene raggiunto sempre dalla catena di chiamate ricorsive?

Assenza di Caso Base

```
function [f] = factRic(n)
  f = n * factRic(n-1);
```

Es. la chiamata a factRic(7)

chiama factRic(6),
che chiama factRic(5),
che chiama factRic(4), ...
che chiama factRic(-1000), ...

- Ottengo una catena infinita di chiamate incondizionate
- Deve sempre esistere una condizione tale per cui non viene eseguita la chiamata ricorsiva (caso base)

Raggiungimento del Caso Base (1)

```
function [f] = factRic(n)
  if (n == 0)
    f = 1;
  else
    f = n * factRic(n-1);
  end
```

Es. la chiamata di factRic(-1)

chiama factRic(-2),
che chiama factRic(-3), ...

- Catena infinita di chiamate incondizionate
- Necessario che questa condizione venga raggiunta

Raggiungimento del Caso Base (2)

```
function [f] = factRic(n)
  if (n == 0)
    f = 1;
  else
    f = n * factRic(n);
  end
```

Es. la chiamata di factRic(7)

chiama factRic(7),
che chiama factRic(7), ...

- Catena infinita di chiamate identiche
- La chiamata ricorsiva non può avere i parametri formali uguali a quelli attuali

Per Finire

- Matlab blocca le chiamate ricorsive dopo un po'

```
>> factRic(600)
Out of memory. The likely cause is an infinite
recursion within the program.
Error in fatRic  at line XXX
```

dove XXX indica la riga in cui compare la chiamata ricorsiva

- Devo valutare se il numero di chiamate ricorsive che mi aspetto è supportato da MatLab

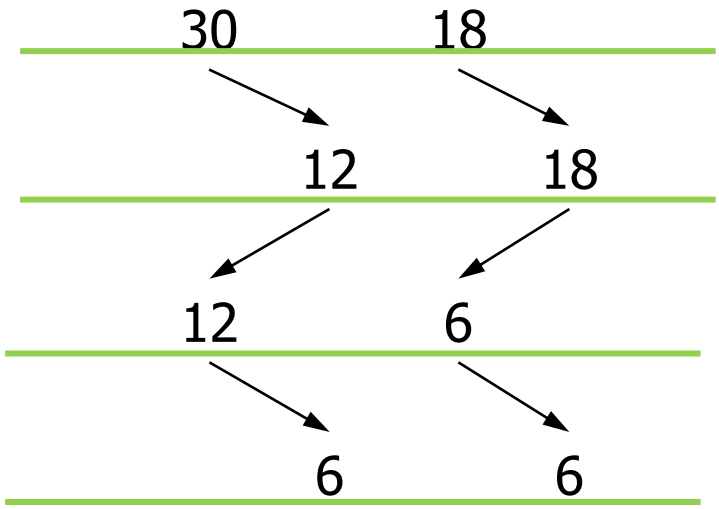
Condizioni Necessarie per una Ricorsione Sicura

- Per evitare ricorsione infinita:
  - Occorre che le **chiamate ricorsive** siano **soggette** a una **condizione** che prima o poi assicura che la catena termini
  - Occorre anche che **l'argomento sia progressivamente ridotto** dal passo induttivo, in modo da tendere prima o poi al caso base
  - Nella pratica l'argomento si avvicina al valore nel caso base

Esempi ed Esercizi (Facoltativi)

Esempio Utile: MCD

- Algoritmo di Euclide
  - se m = n, MCD(m, n) = m (caso base)
  - se m > n, MCD(m, n) = MCD(m-n, n) (caso ricorsivo)
  - se m < n, MCD(m, n) = MCD(m, n-m) (caso ricorsivo)
- Esempio: MCD(30, 18)



Esempio Utile: MCD

- Algoritmo di Euclide
  - se m = n, MCD(m, n) = m (caso base)
  - se m > n, MCD(m, n) = MCD(m-n, n) (caso ricorsivo)
  - se m < n, MCD(m, n) = MCD(m, n-m) (caso ricorsivo)

```
function [M] = MCDeuclidRic(m,n)
  if m == n
      M = m;
  else
      if m > n
          M = MCDeuclidRic(m-n, n);
      else
          M = MCDeuclidRic(m, n-m);
      end
  end
```

Esempio

Scrivere una funzione ricorsiva per calcolare la lunghezza di una stringa

```
function s = calcolaLunghezza(str)
if(isempty(str))
    s = 0;
else
    s = 1 + calcolaLunghezza(str(2
: end));
end
```

Esempio

Scrivere una funzione per stampare una stringa al contrario

```
function stampaAlContrario(frase)

% caso base
if isempty(frase)
    % return
else
    % chiamata ricorsiva
    disp(frase(end)); % stampa al contrario
    stampaAlContrario(frase(1:end-1))
end
```

Esempio

Modificare la funzione per stampare in maniera ricorsiva la stringa nello stesso ordine in cui viene inserita

```
function stampaAlContrario(frase)

% caso base
if isempty(frase)
    % return
else
    % chiamata ricorsiva
     stampaAlContrario(frase(1:end-1))
     disp(frase(end)); % stampa dritto
end
```

NB: In questo caso la ricorsione non è in coda

Esercizio: Cosa Stampa?

```
function stampa(frase)

% caso base
if isempty(frase)
    % return
else
    % chiamata ricorsiva
    stampa(frase(2:end))
    disp(frase(1));
end
```

Esercizio: Cosa Stampa?

```
function stampa(vettore)
% caso base
if (length(vettore) == 1)
    fprintf('%c', vettore(1));
    fprintf('%c', vettore(1));
else
    % chiamata ricorsiva
    fprintf('%c', vettore(1));
    stampa(vettore(2:end));
    fprintf('%c', vettore(1));
end
```

Esercizio: Cosa Stampa?

```
function stampaAlContrario(str)
if(isempty(str))
else
    disp(str(end));
    stampaAlContrario(str(1 : end -1));
    disp(str(end));
end

>> stampaAlContrario('ciao')
      o
      a
      i
      c
      c
      i
      a
      o
```

Esercizio: Cosa Stampa?

```
function stampaAlContrario(str)
if(isempty(str))
else
    disp(str(end));
    stampaAlContrario(str(2 : end));
    disp(str(end));
end

>> stampaAlContrario('ciao')
      o
      o
      o
      o
      o
      o
      o
      o
```

Esercizio: Cosa Stampa?

```
function stampaAlContrario(str)
if(isempty(str))
else
    disp(str(1));
    stampaAlContrario(str(2 : end));
    disp(str(1));
end

>> stampaAlContrario('ciao')
      c
      i
      a
      o
      o
      a
      i
      c
```

Cosa Stampa?

```
function stampaAlContrario(str)
if(isempty(str))
else
    disp(str(1));
    stampaAlContrario(str(1 : end -1));
    disp(str(1));
end

>> stampaAlContrario('ciao’)
      c
      c
      c
      c
      c
      c
      c
      c
```

Esercizio: Cosa Fa?

```
function vet = chefa(vet)
if isempty(vet) || length(vet) == 1
    return;
end
if all(vet(1) >= vet(2 : end))
    return;
end
vet(vet == vet(1)) = [];
vet = chefa(vet);
```

```
>> chefa([19 18 12 19 12 3])

>> chefa([7 18 13 19 12 3])

>> chefa([ 2 3 18 12 19 12 3])
```

Esercizio: Cosa Fa?

```
>> chefa([ 19 18 12 19 12 3])

ans =

    19    18    12    19    12     3

>> chefa([7 18 13 19 12 3])

ans =

    19    12     3

>> chefa([ 2 3 18 12 19 12 3])

ans =

    19
```

Cosa Fa?

- La funzione rimuove restituisce un sottovettore dell'input avente il primo elemento maggiore dei successivi
- La funzione ricerca i sottovettori rimuovendo ad ogni invocazione il primo elemento e tutti i suoi duplicati
- Qualora vi fossero più occorrenze di un elemento diverso dal massimo del vettore, tutte queste vengono rimosse
- Se modificassi la condizione con:

```
all(vet(1) > vet(2 : end))
```

  avremmo che anche il massimo del vettore viene rimosso se questo non è unico

```
>> chefa([ 19 18 12 19 12 3])
     ans = 18    12    12     3
```

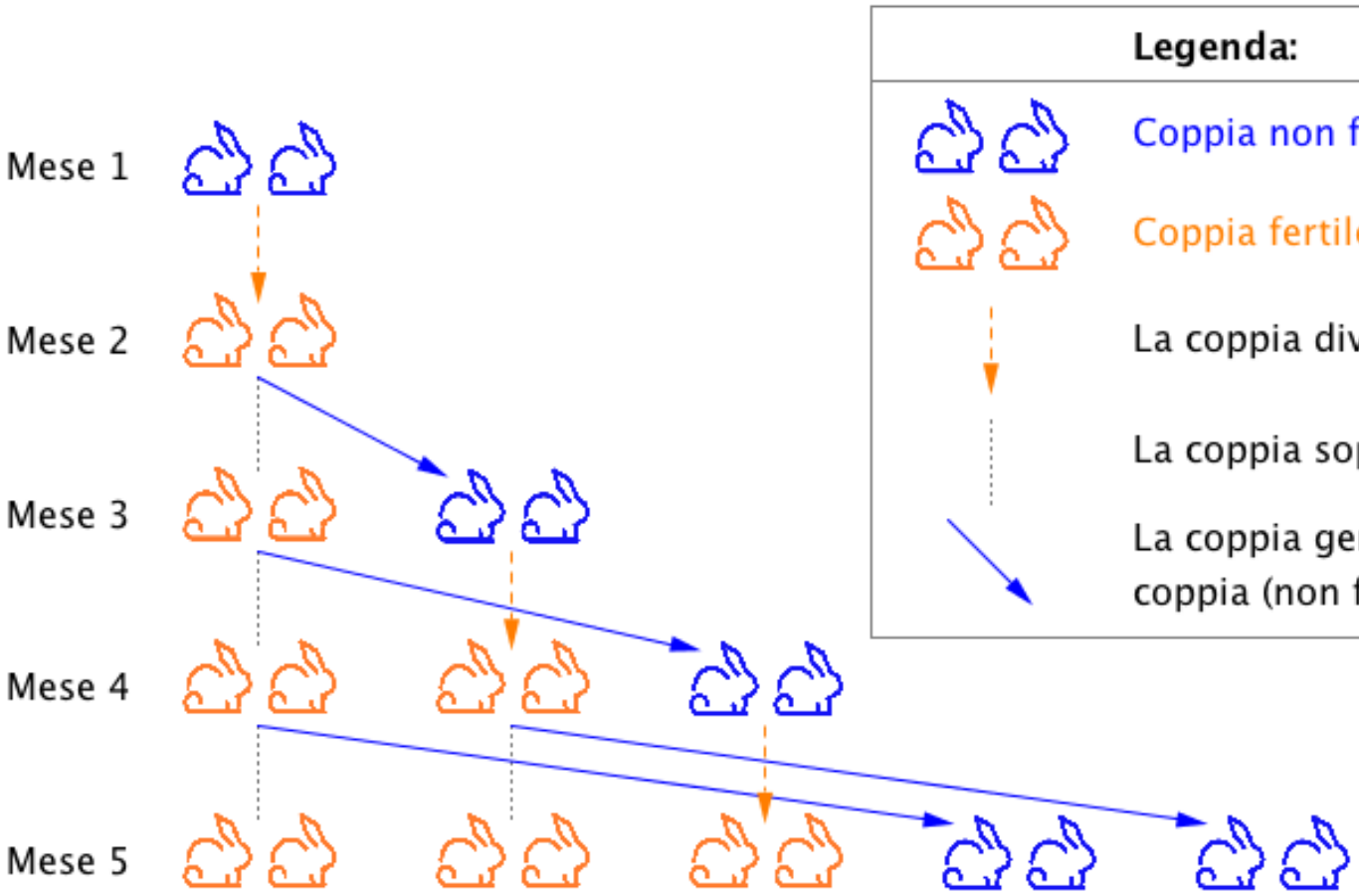
Esempio: la Successione di Fibonacci

Fibonacci (nel 1202) partì dallo studio sullo sviluppo di una colonia di conigli in circostanze ideali:

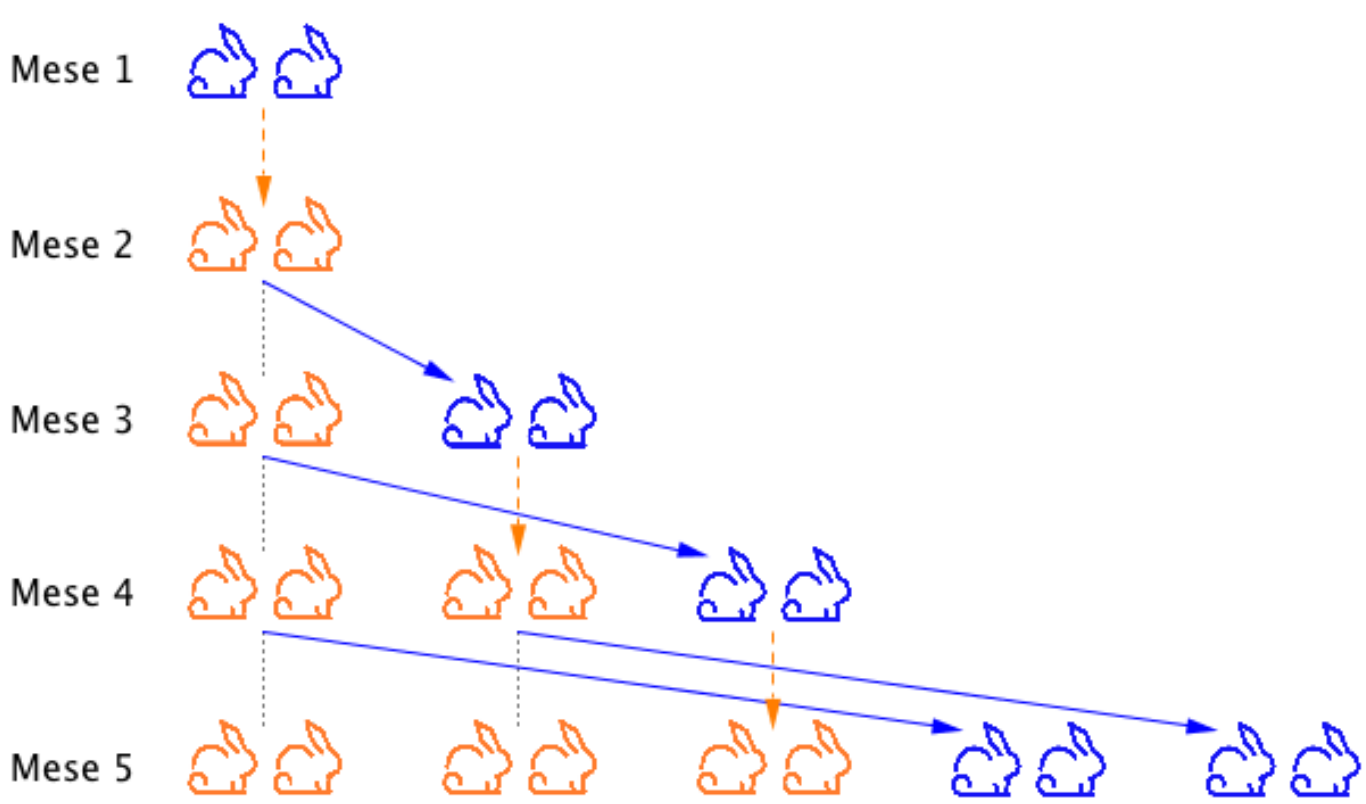
- partiamo da una coppia di conigli
- i conigli possono riprodursi all'età di un mese
- supponiamo che dal secondo mese di vita in poi, ogni femmina produca una nuova coppia
- i conigli non muoiano mai

- Quante coppie ci sono dopo n mesi?

La Successione di Fibonacci Graficamente



La Successione di Fibonacci Graficamente



F(0) = 1

F(1) = 1

F(2) = F(0) + F(1)

F(3) = F(1) + F(2)

F(4) = F(3) + F(2)

Lo Pseudocodice

- È una sequenza di numeri interi in cui ogni numero si ottiene sommando i due precedenti nella sequenza
- I primi due numeri della sequenza sono per definizione pari ad 1

- Se voglio calcolare i numeri di Fibonacci $F = \{f_0, ..., f_n\}$ avrò:
  - $f_0 = 1$
  - $f_1 = 1$
  - Per $n > 1$, $f_n = f_{n-1} + f_{n-2}$

NB: ho due casi base

Codice Ricorsivo Fibonacci

- $f_1 = 1$ (caso base)
- $f_2 = 1$ (caso base)
- $f_n = f_{n-1} + f_{n-2}$ (passo ricorsivo)

```
function [fib] = FiboRic(n)
  if n == 1 | n == 2
      fib = 1;
  else
      fib = FiboRic(n-2) + FiboRic(n-1);
  end
```

Problemi nell'Uso della Ricorsione

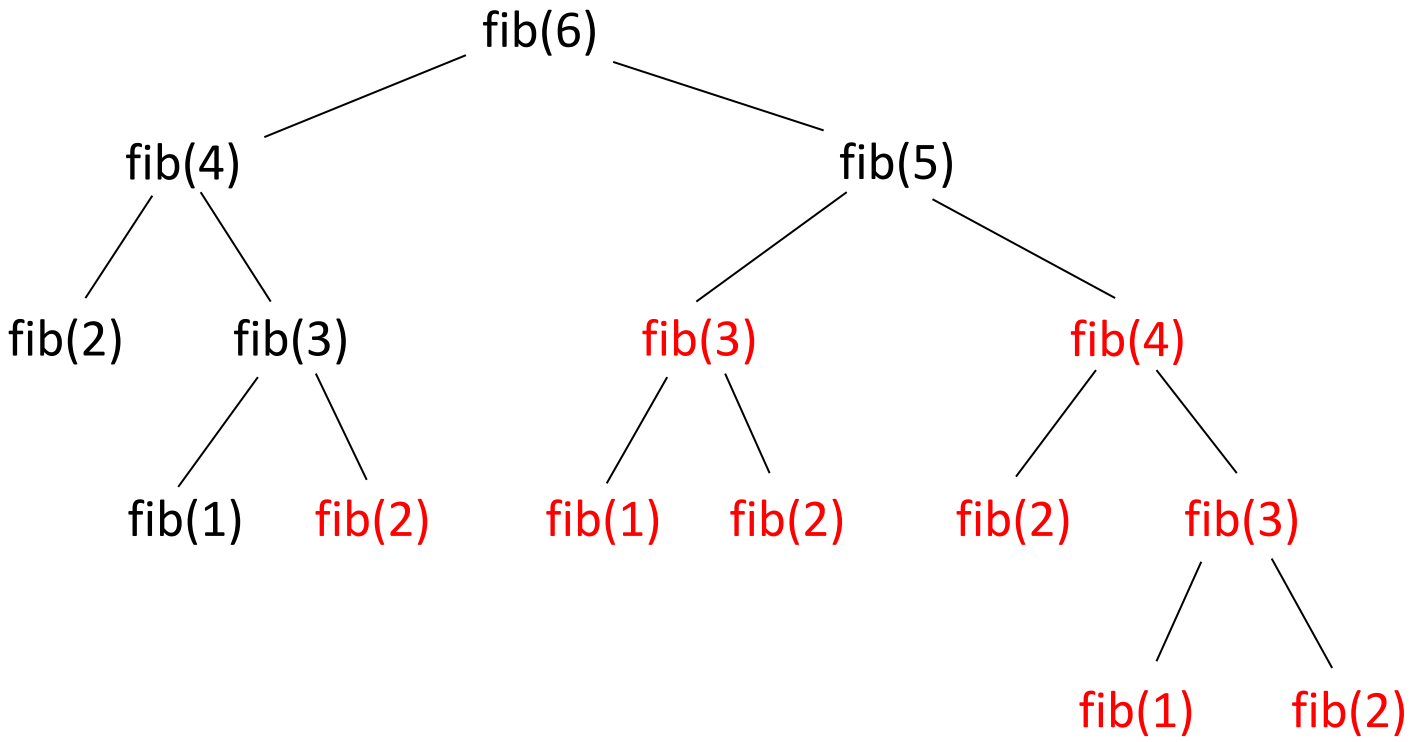
- La programmazione ricorsiva comporta spesso un **uso inefficiente della memoria** per la gestione degli spazi di lavoro delle chiamate generate
- In alcuni casi viene comunque preferita ad altri approcci per la sua eleganza e semplicità
- In altri casi, si può ricorrere ad implementazioni iterative

Es. calcolo dei numeri di fibonacci iterativo

```
function [fl]=Fiblist(n)
fl(1) = 1;
fl(2) = 1;
for k = 3:n
  fl(k) = fl(k-2) + fl(k-1);
end
```

Ricorsione Eccessiva

- Soluzione elegante ma dispendiosa: numero esorbitante di chiamate ricorsive



- Calcolo fib(2), fib(3), fib(4) più di una volta inutilmente

Homework

Scrivere una funzione ricorsiva in Matlab che stampi a schermo tutti i valori del triangolo di Tartaglia per un certo ordine massimo **N** e che restituisca al chiamante la riga **N**-sima

1	N = 0
1 1	N = 1
1 2 1	N = 2
1 3 3 1	N = 3
1 4 6 4 1	N = 4
1 5 10 10 5 1	N = 5
1 6 15 20 15 6 1	N = 6

Palindroma

Scrivere una funzione ricorsiva *palindroma* che controlla se una stringa è palindroma

```
function [res] = controllaPalindromo(stringa)
% stringhe di un carattere (o vuote sono palindrome)
if length(stringa) < 2
    res = true;
else
    % controllo se gli estremi sono uguali
    if stringa(1)==stringa(end)
        % in tal caso richiama controllaPalindromo
su stringa(2, end-1)
        res=controllaPalindromo(stringa(2:end-1));
    else
        res=false;
    end
end
```

Controlla se è Potenza

Scrivere una funzione ricorsiva che controlla se un numero è una potenza di una data base

```
function [res , esp] = isPotenzaDi(num, base)
% caso base
if num == base
  res = 1; esp = 1;
else
  if(mod(num, base) == 0)
    % esp = esp + 1; % da errore
    [res, esp] = isPotenzaDi(num/base, base);
    esp = esp + 1;
  else
    res = 0; esp = NaN;
  end
end
```

Controlla se è Potenza

```
function [res, esp] = isPotenzaDi(num, base)
% caso base
if(num == base)
  res = 1; esp = 1;
elseif(num < base)
  res = 0; esp = NaN;
  % interrompo se n non è intero
elseif(round(num) ~= num)
  res = 0; esp = NaN;
else % chiamata ricorsiva
  [ris, esp] = isPotenzaDi(num/base, base);
  esp = esp +1;
end
```

NB: `res = controllaSePotenza(n, d*d);` è sbagliata perchè la prima volta diventa **`d^2`**, la seconda volta **`d^4`**, etc.

Interpretazione del Codice

```
function r = cosaFa(array)
k = size(array, 2);

if (k == 1)
  r = 1;
elseif (k == 2)
  if (array(1) + array(2) == 10)
      r = 1;
  else
      r = 0;
  end
else
  if (array(1) + array(k) == 10)
      r = cosaFa(array(2:k-1));
  else
      r = 0;
  end
end
```

TDE Febbraio 2012

Si consideri la seguente funzione in Matlab

```
function [ris] = mistero(v, n)
if (n > 1)
    v2 = v(mod(v, n) ~= 0 | v == n);
    ris = mistero(v2, n-1);
else
    ris = v;
end
```

Dire qual è il contenuto di **x** dopo la seguente chiamata

```
x = mistero([2 3 4 5 7 9 11 13 15], 10)
```

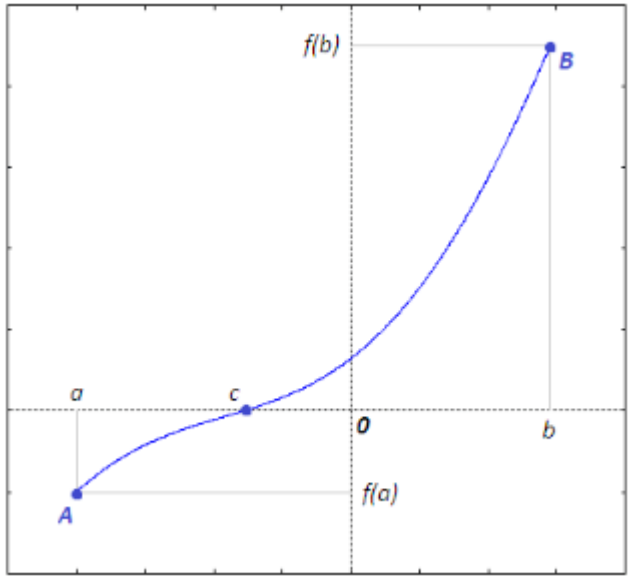
x = 2 3 5 7 11 13

Esempio: il Teorema di Bolzano

Sia f una funzione reale e continua in $[a, b]$ per cui

$$f(a) * f(b) \leq 0$$

allora estiste almeno un punto $c \in [a, b]$ tale che $f(c) = 0$



Usare questa teorema per scrivere una funzione **`haUnoZero`** che prende in ingresso un function handle, gli estremi dell'intervallo e determina se la funzione ha uno zero nell'intervallo

Homework

- Utilizzando il metodo di bisezione e il teorema di Bolzano, scrivere una funzione **ricorsiva e di ordine superiore** **`calcolaZeri`** che calcola gli zeri di una funzione analitica f (passata in un function handle) nell'intervallo continuo $[a, b]$ (gli estremi sono passati come parametri)
- Si consideri come criterio di arresto invece di $f(x) == 0$

  $|f(x)| < \epsilon$ dove ϵ viene passato come parametro