# Struct e Ricorsione

Informatica (ICA) AA 2020 / 2021

Giacomo Boracchi

27 Novembre 2020

giacomo.boracchi@polimi.it

# Strutture in Matlab

# Struct vs Array

Gli **array** permettono di aggregare variabili **omogenee** in una sequenza

Le `struct` permettono di aggregare variabili **eterogenee** in una sola variabile

- Le `struct` è una sorta di "contenitore" per variabili disomogenee di tipi più semplici.
- Le variabili aggregate nella struct sono dette **campi** della struct

*Esempio: variabile per contenere anagrafica di impiegati*

- *nome, cognome, codice fiscale, indirizzo, numero di telefono, stipendio, data di assunzione etc.*
- *Non posso metterli in un array, sono variabili diverse, è molto sconveniente metterle in variabili separate, specialmente se ho diversi impiegati*

# Creazione di una struct

**Creazione di una struttura :**

Utilizzando la funzione struct()

```
studente = struct('nome', 'Giovanni', 'eta', 24)
```

Assegnamento dei valori ai campi (e contestuale definizione dei campi)

```
studente.nome = 'Giovanni';
studente.eta = 24;
```

# Accedere ai campi di una `struct`

Per accedere ai campi si usa l'operatore ***dot***.
Sintassi:

```
nomeStruct.nomeCampo;
```

Quindi, **`nomeStruct.nomeCampo`** diventa, a tutti gli effetti, **una «normale» variabile** del tipo di **`nomeCampo`**.

- **Ai campi** di una struttura applicabili tutte le **operazioni caratteristiche** del tipo di appartenenza
- In questo senso, il *dot* è l'omologo di **`(indice)`** per gli array

# Creazione di una struttura campo per campo

Esempio: creo una struttura studente

```
studente.nome = 'Giovanni Rossi';
studente.indirizzo = 'Via Roma 23';
studente.citta = 'Cosenza';
studente.eta = 25;
```

**Accesso ai campi** come nel C con l'operatore .

**`nomeStruttura.nomeCampo`**

*Es*

```
disp([studente.nome, ' (' , studente.citta ,') ha ' , num2str(studente.eta) , ' anni'])
```

# Creazione di una struttura campo per campo

Esempio: la struttura studente

```
studente.nome = 'Giovanni Rossi';
studente.indirizzo = 'Via Roma 23';
studente.citta = 'Cosenza';
studente.media = 25;
```

É possibile far diventare **studente** un array di strutture, accodando un altro elemento in **studente(2)**.

```
studente(2).nome = 'Giulia Gatti';
studente(2).media = 30;
```

Tutte le strutture dell'array devono avere gli stessi campi (**l'array deve essere omogeneo, la struttura non necessariamente**).

É possibile assegnare solo alcuni campi a **studente(2)**: i campi non assegnati rimangono vuoti.

# Aggiunta di campi

Aggiunta di un campo

```
%facciamo riferiemento alla definizione di studente delle slide precedenti
studente(2).esami = [20 25 30];
```

Il campo esami viene aggiunto a tutte le strutture che fanno parte di studente

- Avrà un valore iniziale per studente(2). Sarà vuoto per tutti gli altri elementi dell'array

# Creazione di una struttura mediante la funzione struct

Consente di preallocare una struttura o un array di strutture

```
S = struct('campo1',val1,'campo2',val2, …)
```

*Es:* 
```
rilieviAltimetrici =
struct('latitudine',30,'longitudine',60, 'altitudine',
1920)
```

# Creazione di una struttura mediante la funzione struct

Consente di preallocare una struttura o un array di strutture

```
S = struct('campo1',val1,'campo2',val2, …)
```

*Es:* 
```
rilieviAltimetrici =
struct('latitudine',30,'longitudine',60, 'altitudine',
1920)
```

Esempio array di strutture:

```
s(5) = struct('x',10,'y',3);
```

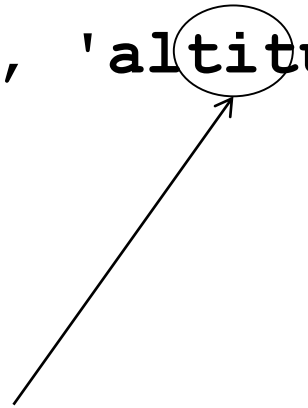
- s è un array 1x5 in cui ogni elemento ha attributi x e y
- solo il quinto elemento di s viene inizializzato con i valori x=10 e y=3
- gli altri elementi vengono inizializzato con il valore di default: [] (array vuoto)

# Creazione di una struttura mediante la funzione struct

Consente di preallocare una struttura o un array di strutture

```
S = struct('campo1',val1,'campo2',val2, …)
```

*Es:* `rilieviAltimetrici(1000) = struct('latitudine',30,'longitudine',[], 'altitudine', 1920)`



Array vuoto. Attenzione: se si Inserisce un valore (es. 20), questo viene assunto dal campo longitudine dell'elemento 1000, ma non dallo stesso campo degli altri elementi dell'array

# Array di strutture innestati

Un campo di una struttura può essere di qualsiasi tipo

E` quindi possibile avere un campo che è, di nuovo, una struttura o un array di strutture

Esempio

```
studente(1).corso(1).nome='InformaticaB';
studente(1).corso(1).docente='Von Neumann';
studente(1).corso(2).nome='Matematica';
studente(1).corso(2).docente='Eulero';
```

corso è un array di strutture

```
>> studente

studente =

    corso: [1x2 struct]
```

# Array di Strutture

In Matlab gli array di strutture vengono gestiti allo stesso modo dell'array numerici e delle stringhe

- È possibile estendere l'array mediante assegnamento
  - Es: `s(7) = s(2);`
- È possibile estrarre sotto-vettori mediante indicizzazione
  - Es `t = s(goodIndexes);`
- È possibile rimuovere elementi da un array di strutture con l'assegnamento al vuoto
  - Es `s(badIndexes) = []`

# Array di Strutture: vincoli

**Attenzione:** valgono i vincoli degli array:

**Tutti gli elementi** di un array di strutture **devono essere omogenei**

In paticolare, **tutte** le strutture nello stesso array devono avere:

- Lo stesso numero di campi
- Tutti i campi con lo stesso nome

(viene tollerato invece un diverso ordinamento dei campi)

```
>> s = struct('a', 10, 'b', 11)
>> t = struct('c', 10, 'a', 11)
>> s(2) = t
Subscripted assignment between dissimilar structures.
```

# Array di Strutture: vincoli

Nota bene: non è necessario che il contenuto dei campi sia dello stesso tipo!

```
s = struct('a', 'pippo', 'b', ones(3))
t = struct('a', ones(3,1), 'b', [])
s(2) = t;
 >> s(1)
    a: 'pippo'
    b: [3×3 double]
>> s(2)
    a: [3×1 double]
    b: []
```

Se necessario quindi si creano campi vuoti (i.e. uguali a `[]`) struttura prima di concatenare una struttura in un array di strutture diverse

# Array di Strutture: particolarità

È possibile accedere rapidamente a tutti i valori di un campo in un array di strutture

```
>> s(1) = struct('a', 'pippo', 'b', 3)
>> s(2) = struct('a', ones(3,1), 'b', 4)
>> s.b
```

L'ultimo comando restituisce

```
ans =
     3
ans =
     4
```

**..e quindi non concatena automaticamente un array!**

# Array di Strutture: particolarità

**Il motivo è che le dimensioni potrebbero non essere consistenti!** Si pensi ad esempio

```
>> s.a
ans =
     'pippo'
ans =
      1
      1
      1
```

Questi non risultano concatenabili!

# Array di Strutture: particolarità

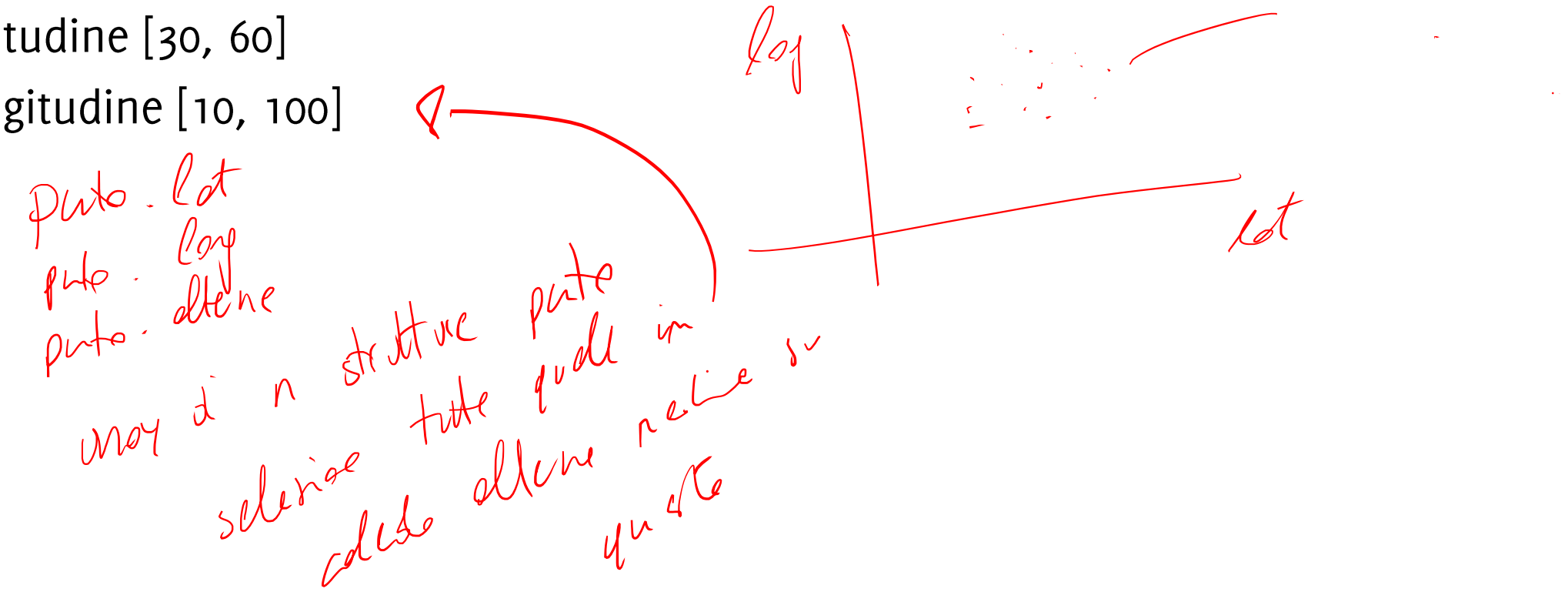
Tuttavia, è possibile **forzare il concatenamento in un array** «a proprio rischio e pericolo», consapevoli che **questo potrebbe sollevare errori**

```
>> v = [s.b]

v =

     3     4

>> v = [s.a]

v =

     3     4

Error using horzcat

Dimensions of matrices being concatenated are not
consistent.
```

# Esercizio

Si sviluppi uno script matlab che acquisisce da tastiera i dati relativi ad un numero arbitrario di rilievi altimetrici e che quindi stampa a video l'altitudine media di tutti i rilievi che si trovano nell'intervallo

- latitudine [30, 60]
- longitudine [10, 100]



# Esercizio, la roulette

```
% Scrivere un programma per simulare il gioco della roulette
% la roulette possiede 38 numeri (da 1 a 36, lo zero e il doppiozero)
% 0 e 00 non sono ne pari ne dispari (vince il banco)
%
% il banco inizialmente possiede 5000 euro
% i giocatori possiedono inizialmente 5000 euro
%
% 1) assumere ad ogni giocata che il giocatore 1 punti 5 euro su pari
% o dispari con la stessa probabilità
% se vince, giocatore1, ottiene 2 volte la posta,
% se perde il banco incassa il valore giocato.
%
%   Mostrare la variazione dell'ammontare del banco e del
%   giocatore all'aumentare delle giocate fino a che o il
%   giocatore perde il banco viene sbancato
%
```

% hints
% - utilizzare la funzione rand() per generare numeri uniformemente distribuiti in [0,1]. Riscalarli quindi in [0 , 38] e approssimarli
% - utilizzare un array di strutture per contenere i giocatori (è possibile aggiungere ulteriori campi alle strutture)
% - utilizzare, dove possibile, funzioni da voi sviluppate

```
%
%2) aggiungere un secondo giocatore che punta sempre 1 euro
sul 15 (se esce 15 vince 36 volte la posta)
%
%3) aggiungere un terzo giocatore che usa la seguente strategia:
%   egli punta sempre sul pari e inizialmente punta un euro.
%   se vince ricomincia a puntare un euro sempre sul pari
%   se perde raddioppia la puntata sempre sul pari,
%   se non ha abbastanza soldi punta tutto quello che possiede
%
```

# Richiami sulle Funzioni

# Un esempio

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

# Un esempio

Parametro formale in uscita

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

Parametro formale in ingresso

**Definizione di $n!$**

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

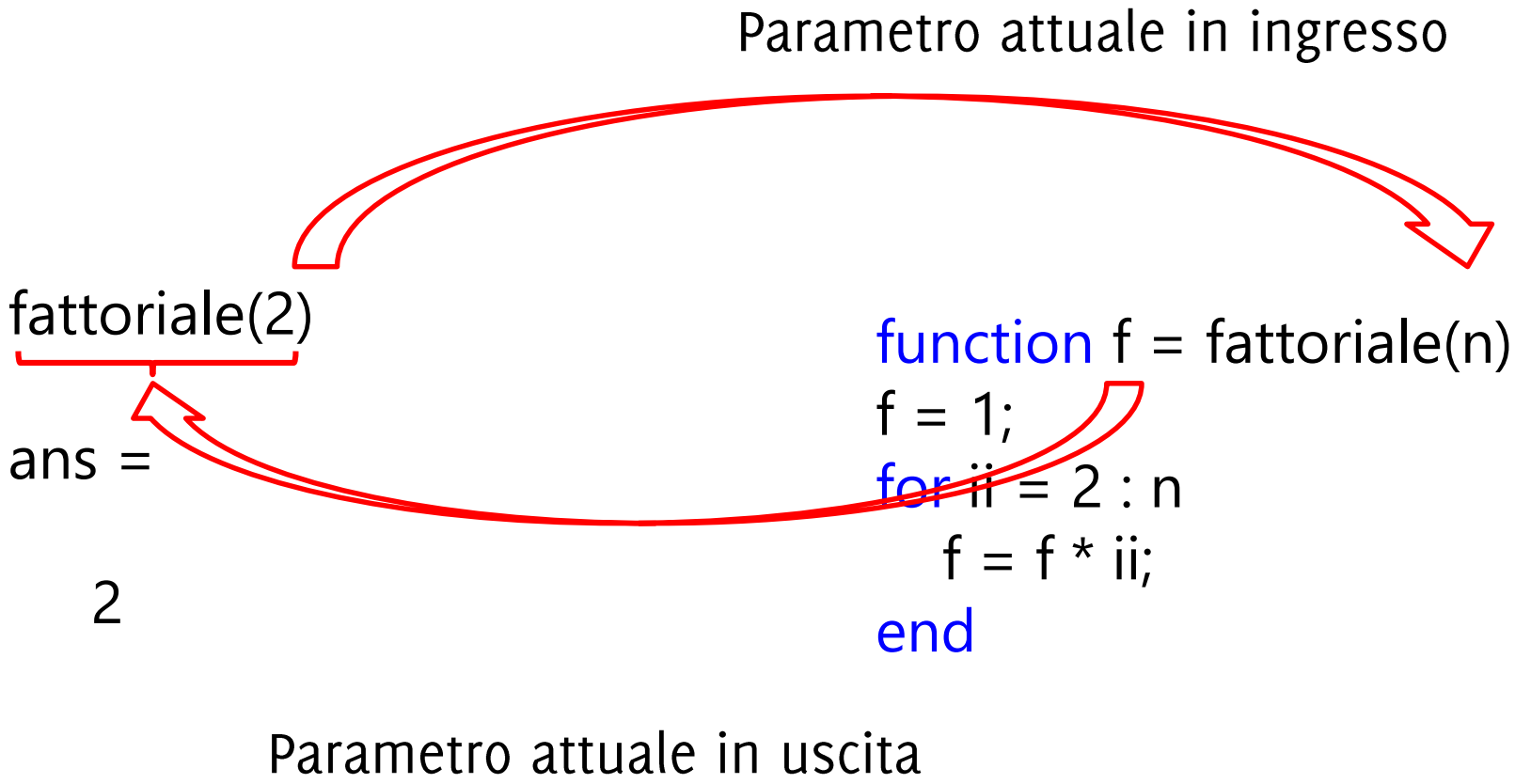
$$n! = n * (n-1) * (n-2) * \cdots * 2 * 1$$

# Un esempio di chiamata

```
fattoriale(2)

ans =

     2
```

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

# Un esempio di chiamata



# 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

# 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 n, f,ii
- Non ha visibilità su k ed f2
- Non ha legami con il workspace principale se non per i parametri attuali che vengono copiati (sia in ingresso che in uscita)
- Distrutto terminata l'esecuzione

# Un esempio

f2 = fattoriale(2)

f2 =

2

f3 = fattoriale(3)

f3 =

6

f4 = fattoriale(4)

f4 =

24

f5 = fattoriale(5)

f5 =

120

f6 = fattoriale(6)

f6 =

720

# Un esempio

Vale la seguente relazione

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

si dimostra immediatamente dalla definizione di fattoriale

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

$$(n-1)!$$

calcolaFatt(n) → n · calcolaFatt(n-1)

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

7! = 7 · 6!

6 · 5 · 4 · 3 · 2 · 1

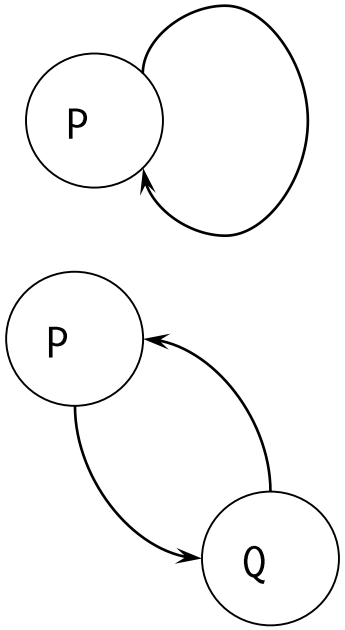
# Ricorsione

Programmi che chiamano se stessi

# 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**

# Esempio: il fattoriale Ricorsivo

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

# Esempio: il fattoriale Ricorsivo

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

Questa ci ricorda $0! = 1$

# Esempio: il fattoriale Ricorsivo

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

Questa ci ricorda

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

Una funzione che chiama se stessa

# Una Funzione che Chiama se Stessa?

In ogni istante possono essere in corso **diverse attivazioni** dello **stesso** sottoprogramma

- Ovviamente sono **tutte sospese tranne** una, **l'ultima invocata**, all'interno della quale si sta svolgendo il **flusso di esecuzione**.

# Una Funzione che Chiama se Stessa?

In ogni istante possono essere in corso **diverse attivazioni** dello **stesso** sottoprogramma

- Ovviamente 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** (in Matlab, ogni funzione attivata ha un workspace distinto)

- Si hanno quindi **copie distinte** dei **parametri attuali** e delle **variabili locali** nelle varie invocazioni

# Una funzione che chiama se stessa?

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

# Una funzione che chiama se stessa?

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

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:
f(n) = n! = n*(n-1)*(n-2)*…*3*2*1

Definire caso base, e passo ricorsivo

# Esempio: il fattoriale

Definizione:

f(n) = n! = n*(n-1)*(n-2)*…*3*2*1

$n \cdot (n-1)!$

Passo ricorsivo:

f(n) = n*f(n-1)

Caso base:

f(0)=1

# Una funzione che chiama se stessa?

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

# Una funzione che chiama se stessa?

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

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:
f(n) = n! = n*(n-1)*(n-2)*…*3*2*1

Definire caso base, e passo ricorsivo

# Esempio: il fattoriale

Definizione:
$f(n) = n! = n*(n-1)*(n-2)*\ldots*3*2*1$

Passo ricorsivo:
$f(n) = n*f(n-1)$

Caso base:
$f(0)=1$

# Esempio: il fattoriale

Definizione:
f(n) = n! = n*(n-1)*(n-2)*…*3*2*1

Passo ricorsivo:
f(n) = n*f(n-1)

Caso base:
f(0)=1

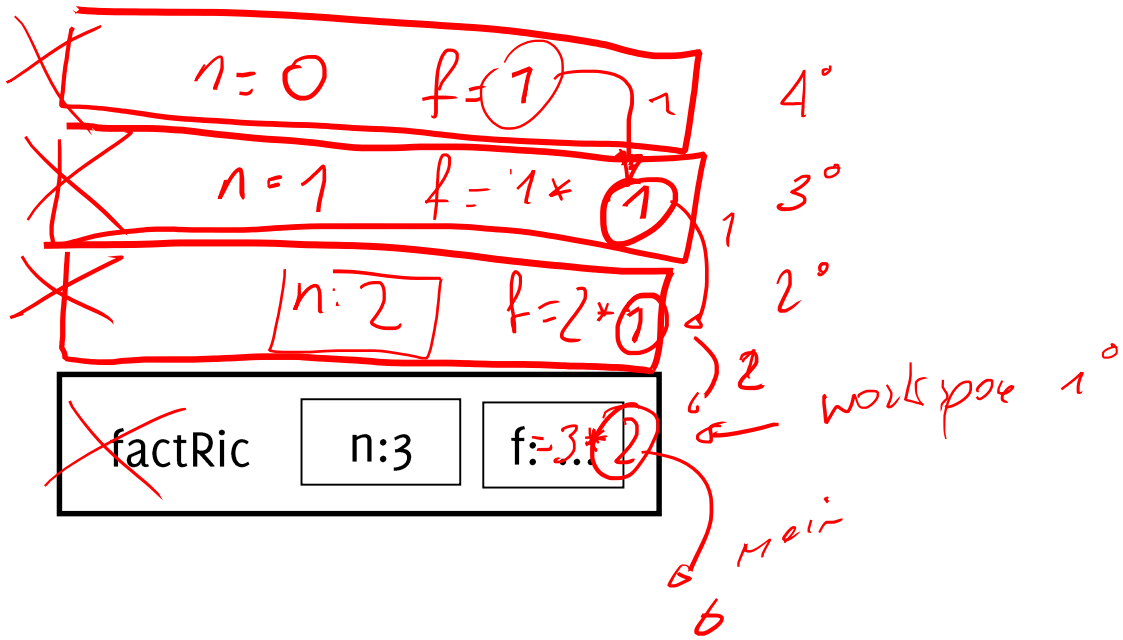
```
function [f]=factRic(n)
  if (n==0)
      f=1;
  else
      f=n*factRic(n-1);
  end
```

# Funzionamento ricorsione

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

caso base

**factRic(3)**

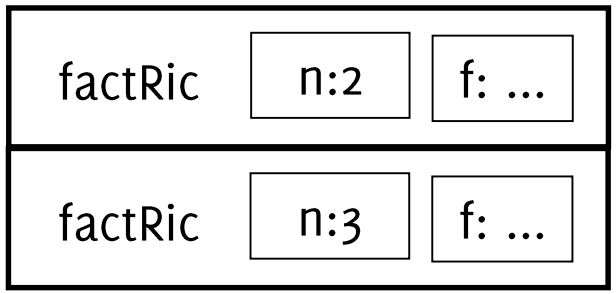



ans = 6

# Funzionamento ricorsione

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

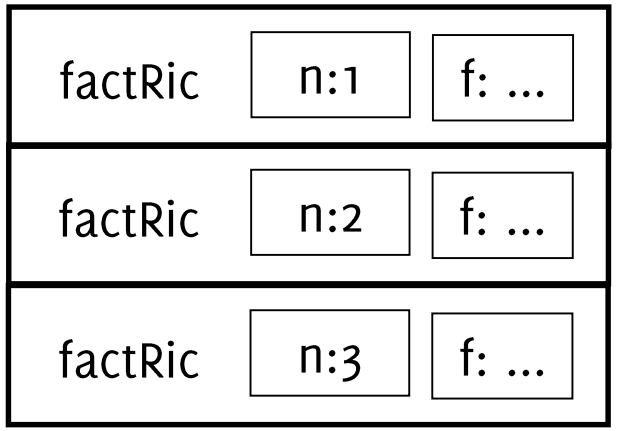
**factRic(3)**



# Funzionamento ricorsione

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

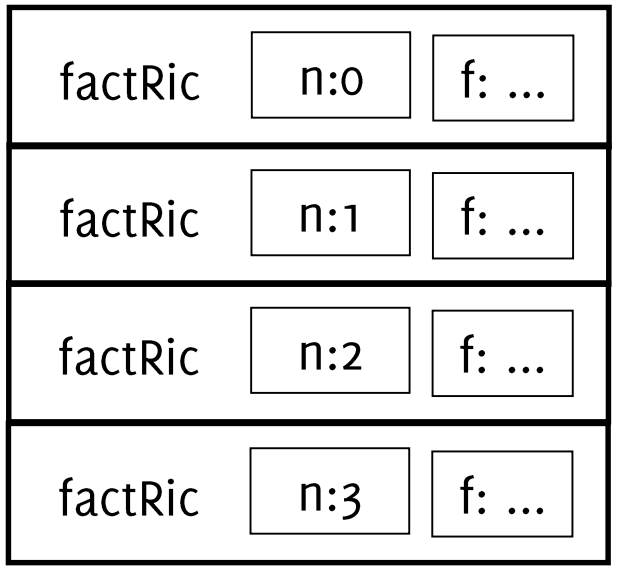
**factRic(3)**



# Funzionamento ricorsione

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

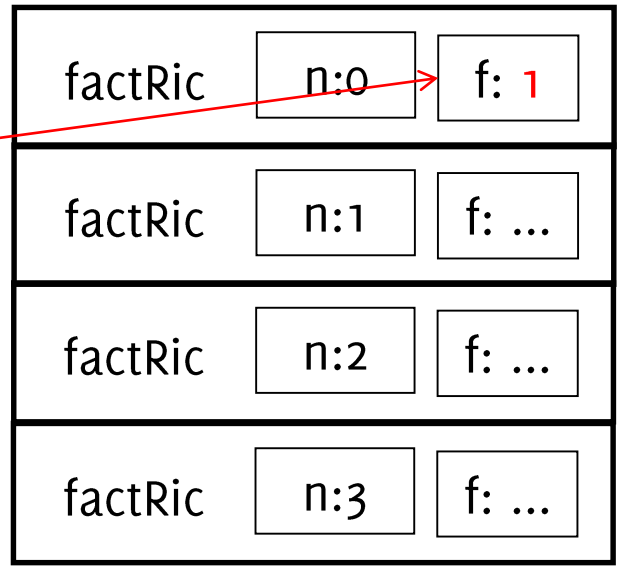
**factRic(3)**



# Funzionamento ricorsione

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

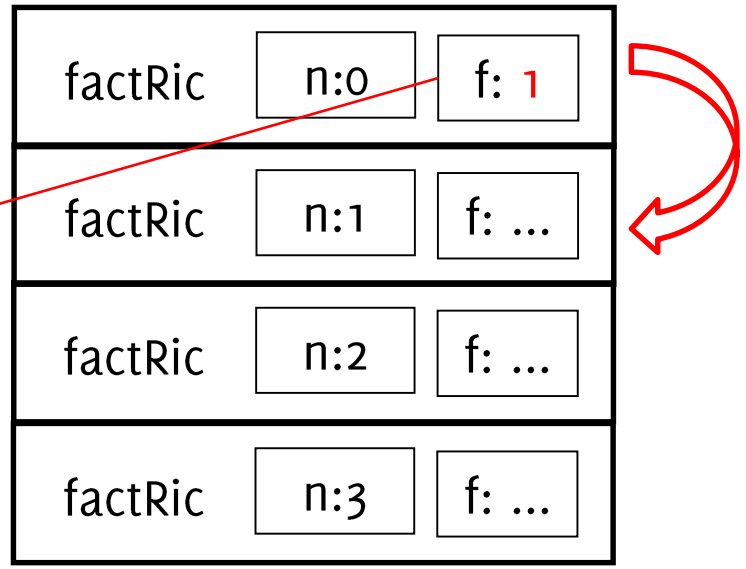
**factRic(3)**



# Funzionamento ricorsione

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

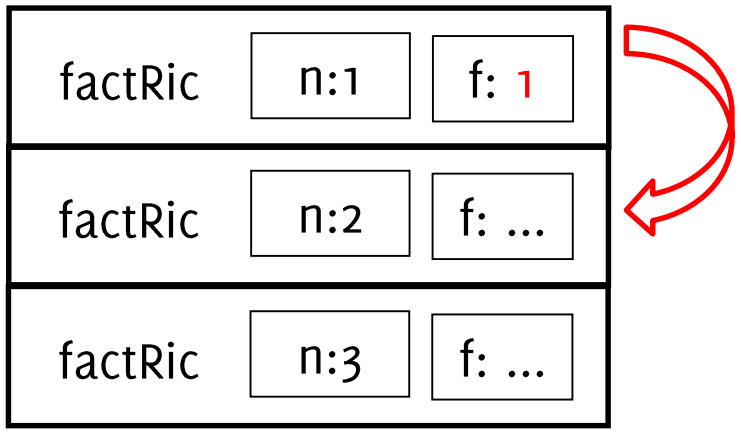
**factRic(3)**



# Funzionamento ricorsione

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

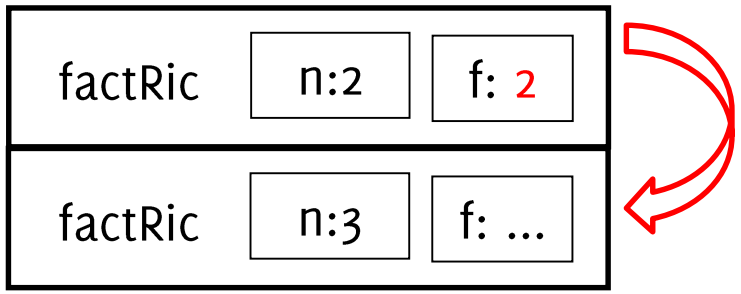
**factRic(3)**



# Funzionamento ricorsione

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

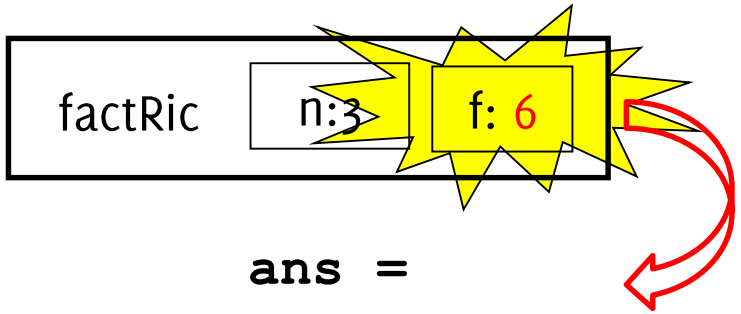
**factRic(3)**



# Funzionamento ricorsione

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

**factRic(3)**




**ans =**

**6**

# Ricorsione in Coda

Ricorsione in cui la funzione:

- 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
```

# Problemi nell'Uso della Ricorsione

Terminazione della catena ricorsiva

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

# Errori nell'Uso della Ricorsione

**Catena infinita di chiamate incondizionate**

- **Deve esistere** una **condizione** tale per cui **non** viene eseguita la **chiamata ricorsiva** (**caso base**)

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

Es la chiamata a factRic(7) chiama factRic(7),
che chiama factRic(6),
che chiama factRic(5),
che chiama factRic(4),
che chiama factRic(3),….

che chiama factRic(-1000),….

# Errori nell'Uso della Ricorsione

**Catena infinita di chiamate incondizionate**

- Attenzione che **è necessario che questa condizione venga raggiunta**: anche programmi formalmente corretti potrebbero non funzionare per alcuni valori degli ingressi

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

- Ad es, factRic(-1) da luogo ad una sequenza di chiamate infinite

# Errori nell'Uso della Ricorsione

**Catena infinita di chiamate identiche:**

- La chiamata ricorsiva **non** può avere i **parametri formali uguali a quelli attuali**

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

Es la chiamata a factRic(7) chiama factRic(7),
che chiama factRic(7),
che chiama factRic(7),
che chiama factRic(7),
che chiama factRic(7),….

# Di Default…

Matlab si blocca a 500 chiamate ricorsive

>> factRic(600)

Out of memory. The likely cause is an infinite recursion within the program.

Error in **fatRic** at line XXX

# Condizioni Necessarie

.. 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

# Esempio

Scrivere una funzione ricorsiva che calcola la somma di tutti gli interi compresi tra i due argomenti

```
funtia S = SommaCompresi(a, b)
if (a == b)
    S = a;
    return;
end
S = a + SommaCompresi(a+1, b);
```

7 8 9 10 11

a + SommaCompresi(a+1, b)

b + a + SommaCompresi(a+1, b-1)


# Esempio

Scrivere una funzione ricorsiva che calcola la somma di tutti gli interi compresi tra i due argomenti

```
function somma = sommaNumeriCompresi(a ,b)

if a == b
    somma = a;
    disp(['caso base somma vale ' , num2str(somma)]);

else
    disp(['prima chiamata a = ' ,num2str(a)]);
    somma = a + sommaNumeriCompresi(a+1 , b);
    disp(['dopo chiamata a = ' ,num2str(a)]);
end
```

# Esempio

Scrivere una funzione ricorsiva che calcola la somma di tutti gli interi compresi tra i due argomenti

```
function somma = sommaNumeriCompresi(a ,b)

if b< a
    somma = 0;
end
if a == b
    somma = a;
    disp(['caso base somma vale ' , num2str(somma)]);
else
    disp(['prima chiamata a = ' ,num2str(a)]);
    somma = a + sommaNumeriCompresi(a+1 , b);
    disp(['dopo chiamata a = ' ,num2str(a)]);
end
```

Errore! Non è un caso base perchè dopo di questo viene comunque eseguita la chiamata ricorsiva

# Esempio

Scrivere una funzione ricorsiva che calcola la somma di tutti gli interi compresi tra i due argomenti

```
function somma = sommaNumeriCompresi(a ,b)

if b< a
    somma = 0;
    return;
end
if a == b
    somma = a;
    disp(['caso base somma vale ' , num2str(somma)]);
else
    disp(['prima chiamata a = ' ,num2str(a)]);
    somma = a + sommaNumeriCompresi(a+1 , b);
    disp(['dopo chiamata a = ' ,num2str(a)]);
end
```

In questo moto l'esecuzione termina e abbiamo un caso base corretto. Si sarebbe ottenuto lo stesso risultato annidando gli if o utilizzando variabili di flag

# Esempio

Scrivere una funzione ricorsiva che calcola la somma di tutti gli interi compresi tra i due argomenti

```
function somma=calcolaSommaCompresi2(a_temp,b_temp)
a=min([a_temp,b_temp]);
b=max([a_temp,b_temp]);
if(a==b)
    disp(['caso base 1 a=',num2str(a),'b=',num2str(b)])
    somma=a;
else
    if(b-a==1)
        disp(['caso base 2 a=',num2str(a),'b=',num2str(b)])
        somma=a+b;
    else
        disp(['prima della chiamata ricorsiva a=',num2str(a),'b=',num2str(b)])
        somma=a+calcolaSommaCompresi2(a+1,b-1)+b;
        disp(['dopo la chiamata ricorsiva a=',num2str(a),'b=',num2str(b),'
somma =',num2str(somma)])
    end
end
```

Inverte le variabili per calcolare comunque i numeri compresi anche se l'ordine non è corretto

# Esempio

Scrivere una funzione ricorsiva che calcola la somma di tutti gli interi compresi tra i due argomenti

```
function somma = sommaCompresi(a, b)
% calcolare la somma degli interi tra a e b

if(b < a)
    somma = 0;
    return
end

if (a == b)
    somma = a;
else
    somma = a + sommaCompresi(a + 1, b - 1) + b;
end
```

In questo caso restituisce 0 se $b < a$ e non serve un caso base per il caso $b - a == 1$

# Esempio

Scrivere una funzione ricorsiva per calcolare la lunghezza di una stringa

# 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
```

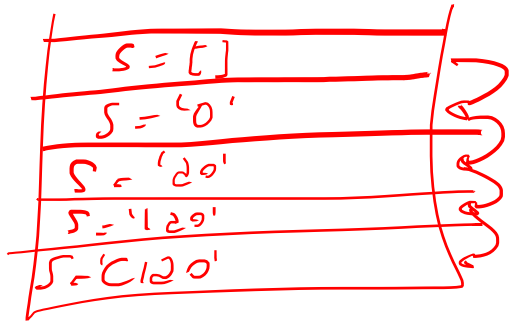
# Esempio:

Scrivere una funzione ricorsiva per stampare una stringa al contrario

```
if isempty(stringa)
    return;
end
stampaContr(stringa(2:end));
disp(stringa(1));
```

```
disp(stringa(end))
stampaContr(stringa(1:end-1))
```


o 2 i c



# 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 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
```

In questo caso la ricorsione non è in coda

# Esempio:

Cosa stampa??

```
function stampa(frase)

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

# Esempio:

Cosa stampa??

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

# Cosa Stampa?

```
function stampaAlContrario(str)

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

# Cosa Stampa?

```
function stampaAlContrario(str)

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

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

# Cosa Stampa?

```
function stampaAlContrario(str)

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

# Cosa Stampa?

```
function stampaAlContrario(str)

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

```
›› stampaAlContrario('ciao')
     o
     o
     o
     o
     o
     o
     o
     o
     o
```

# Cosa Stampa?

```
function stampaAlContrario(str)

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

# Cosa Stampa?

```
function stampaAlContrario(str)

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

# Cosa Stampa?

```
function stampaAlContrario(str)

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

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