Struct e Ricorsione

Informatica, AA 2021/2022

Francesco Trovò

19 Novembre 2021

https://trovo.faculty.polimi.it/

francesco1.trovo@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*
- *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

- Primo modo: utilizzando la funzione struct()

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

in generale:

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

- Secondo modo: 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: 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

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

Array di strutture

Esempio:

```
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)**

Array di strutture innestati

Un campo di una struttura può essere di qualsiasi tipo

E` quindi possibile avere un campo che è a sua volta 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 particolare, **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', 'bilbo', 'b', ones(3))
t = struct('a', ones(3,1), 'b', [])
s(2) = t;
 >> s(1)
    a: 'bilbo'
    b: [3×3 double]
>> s(2)
    a: [3×1 double]
    b: []
```

Se necessario quindi si creano campi vuoti (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', 'bilbo', '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
```

i quali 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 (1)

```
% 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
%
```

Esercizio, la roulette (2)

% 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

Esercizio, la roulette (3)

```
%
%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

Parametro formale in uscita

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

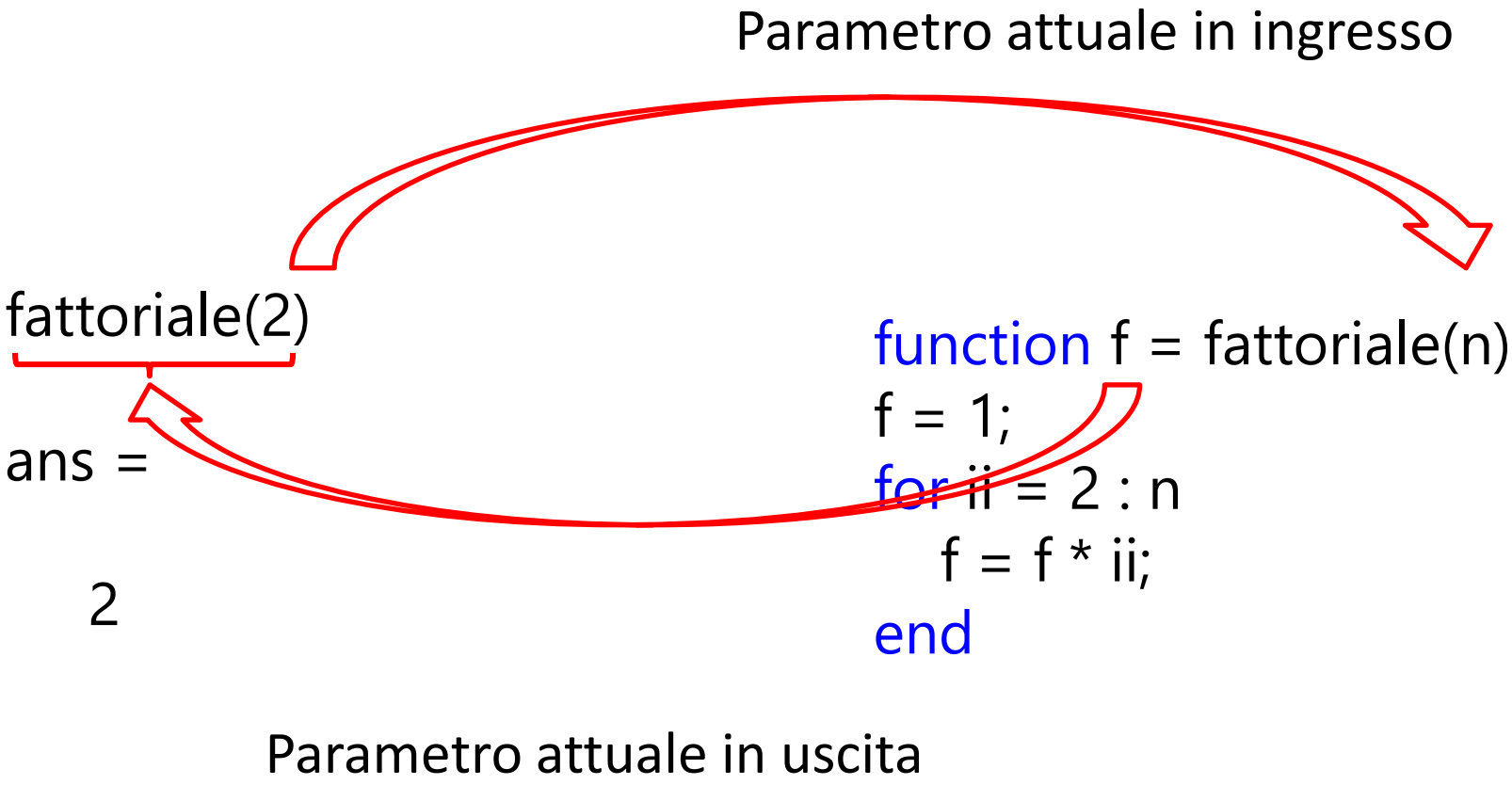
Parametro formale in ingresso

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



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

Nota sul calcolo del fattoriale

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?

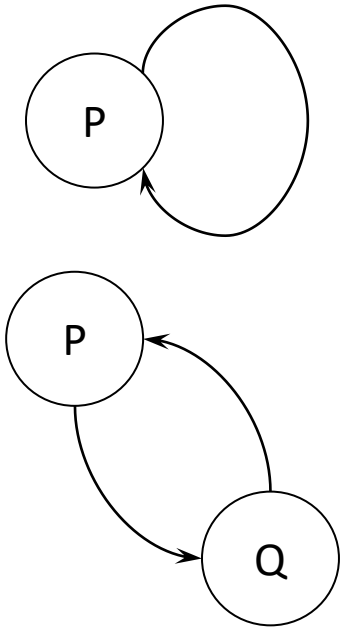
Ricorsione

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**

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

… 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

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

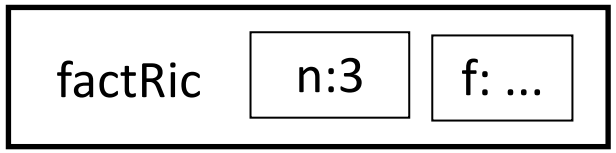
Caso base:
f(0)=1

```
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
```

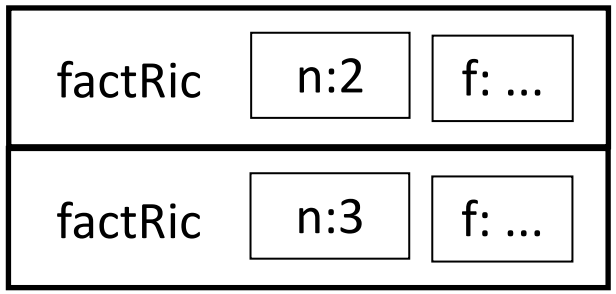
factRic(3)



Funzionamento ricorsione

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

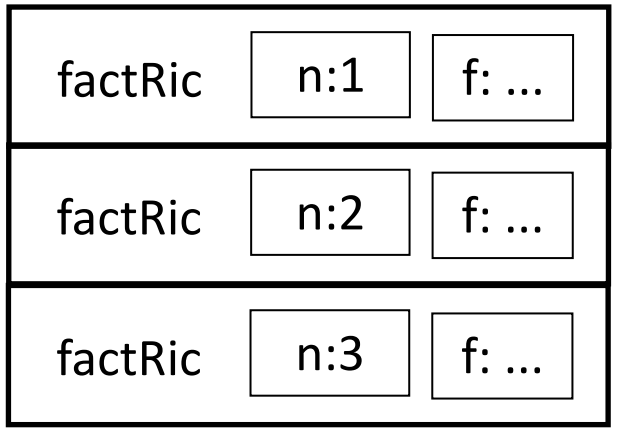
factRic(3)



Funzionamento ricorsione

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

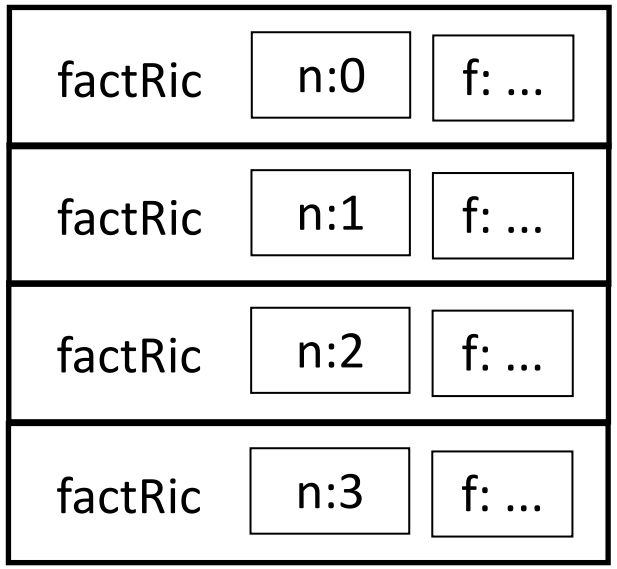
factRic(3)



Funzionamento ricorsione

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

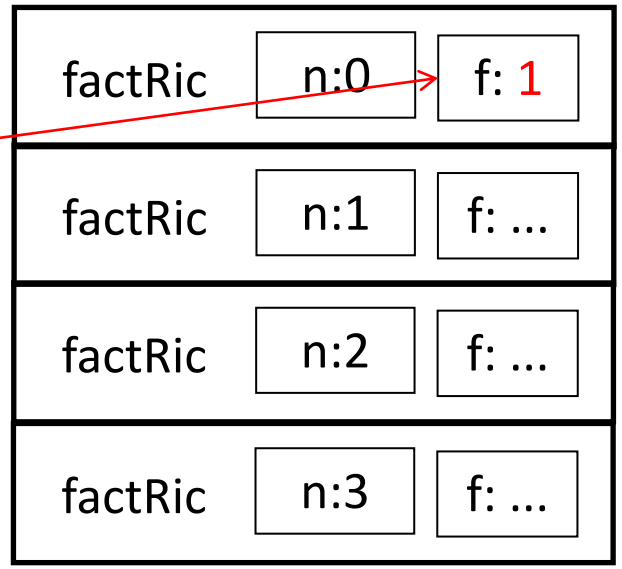
factRic(3)



Funzionamento ricorsione

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

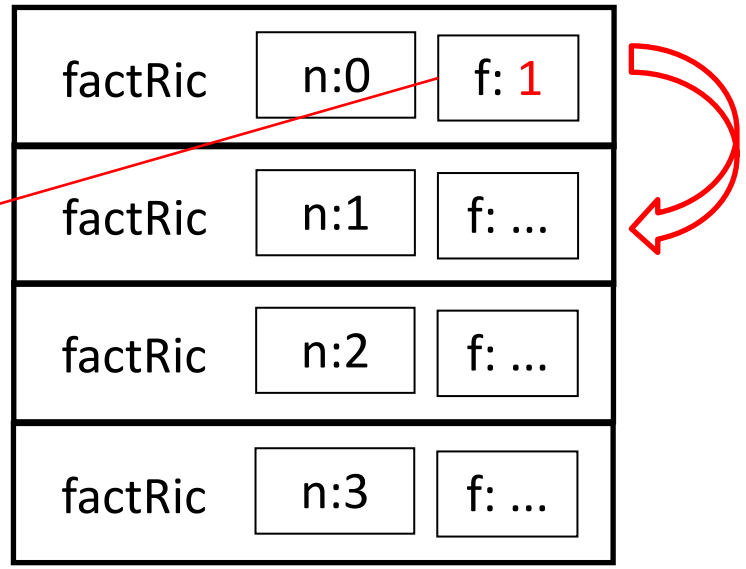
factRic(3)



Funzionamento ricorsione

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

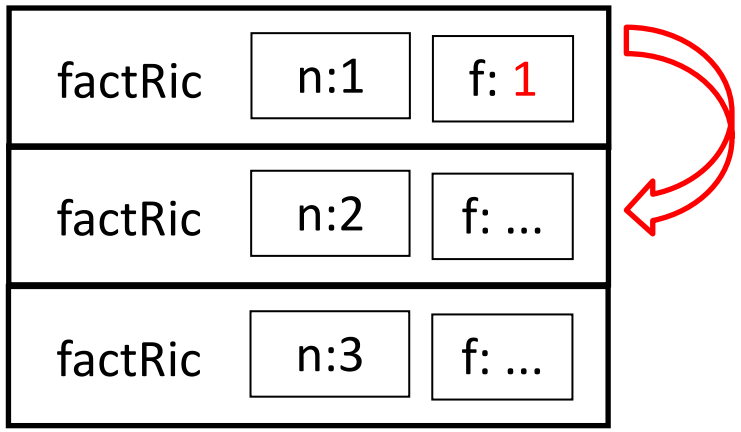
factRic(3)



Funzionamento ricorsione

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

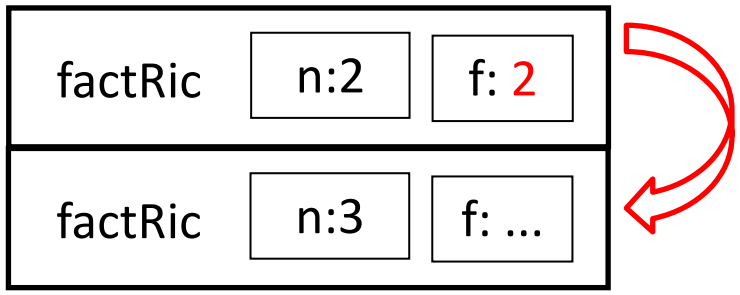
factRic(3)



Funzionamento ricorsione

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

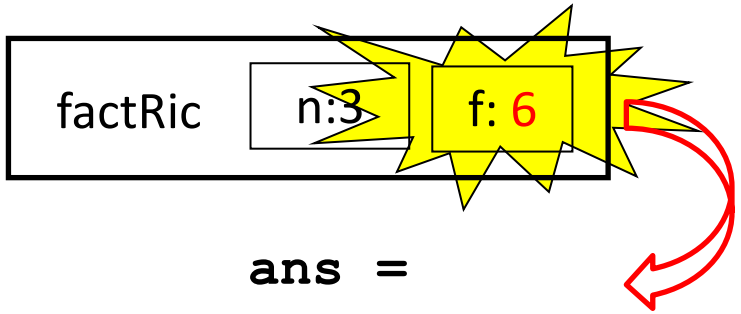
factRic(3)



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
```

Errori Comuni con la Ricorsione

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),….

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)

Esempi di Ricorsione

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 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 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

Esercizio:

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
```

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
```

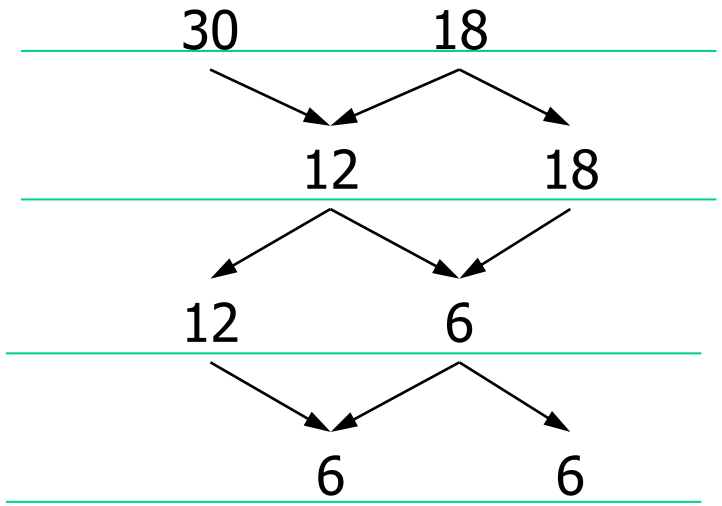
```
>> stampaAlContrario('ciao')
    c
    c
    c
    c
    c
    c
    c
    c
```

Esempio: MCD

Algoritmo di Euclide

- se m = n, MCD(m,n) = m (caso base)
- se m > n, MCD(m,n) = MCD(m-n,n) (caso risorsivo)
- se m < n, MCD(m,n) = MCD(m,n-m) (caso risorsivo)

Esempio: MCD(30,18)



Esempio: MCD

Algoritmo di Euclide

- se m = n, MCD(m,n) = m (caso base)
- se m > n, MCD(m,n) = MCD(m-n,n) (caso risorsivo)
- se m < n, MCD(m,n) = MCD(m,n-m) (caso risorsivo)

```
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
```

Due possibili chiamate ricorsive, e la chiamata è condizionata

Un altro esempio: la serie di Fibonacci

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

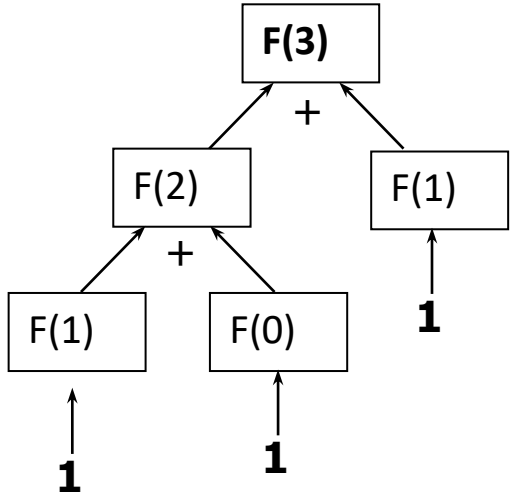
- Partiamo da una coppia di conigli
- I conigli possono riprodursi all'età di un mese
- Supponiamo che dal secondo mese di vita in poi, ogni femmina produca una nuova coppia
- e inoltre che i conigli non muoiano mai…

- Quante coppie ci sono dopo n mesi?

Definizione ricorsiva della serie

I numeri di Fibonacci

- Modello a base di molte dinamiche evolutive delle popolazioni



$F = \{f_0, \ldots, f_n\}$

- $f_0 = 1$
- $f_1 = 1$

} casi base (sono 2!)

- Per $n > 1$, $f_n = f_{n-1} + f_{n-2}$

} 1 passo induttivo

Notazione "funzionale": $F(i) = f_i$

Esempio: Fibonacci

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.

- $f_1 = 1$ (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

Uso della memoria

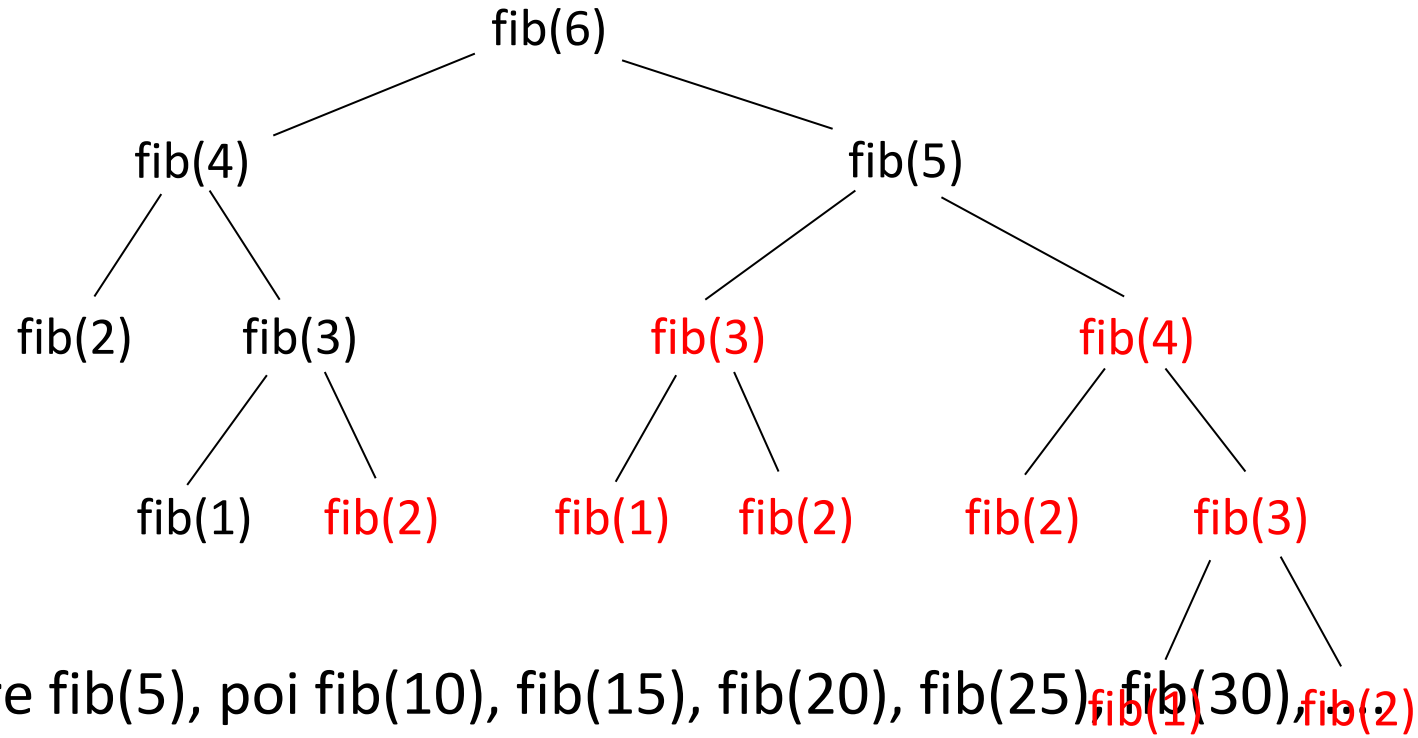
- La programmazione ricorsiva comporta spesso un uso inefficiente della memoria per la gestione degli spazi di lavoro delle chiamate generate
- In alcuni casi viene comunque preferita ad altri approcci per la sua eleganza e semplicità
- In altri casi, si può ricorrere ad implementazioni iterative
- Esempio

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

Funzione iterativa che calcola i primi n numeri di fibonacci

Ricorsione Eccessiva

Soluzione elegante ma dispendiosa: numero esorbitante di chiamate ricorsive



Provate a far calcolare fib(5), poi fib(10), fib(15), fib(20), fib(25), fib(30), ...

Qante volte viene calcolato fab(3)????

Meglio usare una soluzione non ricorsiva...

TODO

Scrivere una funzione ricorsiva in Matlab che stampi a schermo tutti i valori del triangolo di Tartaglia per un certo ordine massimo N. i.e., 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
```

Le ninfee

Uno stagno è pieno di ninfee

- Ogni ninfea in 1 giorno si riproduce, generando un’altra ninfea
- Dopo 30 gg lo stagno è pieno

Quanto hanno impiegato le ninfee a riempire a metà stagno?

Le ninfee

Uno stagno è pieno di ninfee

- Ogni ninfea in 1 giorno si riproduce, generando un'altra ninfea
- Dopo 30 gg lo stagno è pieno

Quanto hanno impiegato le ninfee a riempire a metà stagno?

Scrivere una funzione ricorsiva per modellare

- una popolazione iniziale di *n* ninfee,
- che si riproduce ad un fattore *f* ogni giorno,

e per dire quanti giorni richiede per raggiungere una popolazione di *N* individui

Soluzione

```
function gg = stagno(n, f, N)
 % caso base: se ho già N ninfee nello stagno
% non devo aspettare alcun giorno
if (n >= N)
    gg = 0;
else
    % non abbiamo ancora N ninfee -> chiamata ricorsiva:
    % il numero di giorni necessari per coprire lo stagno è
    % uguale a: un giorno +
    % il numero di giorni che saranno necessari domani
    % (quando le ninfee saranno diventate n*f).
     gg = 1 + stagno(n * f, f, N);
end
```

Ricorsione:

Scrivere una **funzione ricorsiva** che prende in ingresso due numeri n e d che determina se n è una potenza di d

Soluzione

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

Altra soluzione

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

Altra soluzione

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

N.B.
res = controllaSePotenza(n,d*d);
è sbagliata perchè
la prima volta diventa d^2 la seconda volta d^4...

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
```

1. Dire qual è il contenuto di x dopo la seguente chiamata

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

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
```

Elementi di v alle posizioni dove v non è multiplo di n oppure dove v è uguale ad n

1. Dire qual è il contenuto di x dopo la seguente chiamata

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

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
```

Elementi di v alle posizioni dove v non è multiplo di n oppure dove v è uguale ad n

1. 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]
```

2. Implementare un frammento di programma in linguaggio Matlab che legga da tastiera un numero intero positivo A. Quindi, chiami la funzione mistero passandole, come primo argomento, un vettore contenente tutti i numeri interi compresi fra 2 e A e, come secondo argomento, il numero A.
   Infine, stampi a video il valore ritornato dalla chiamata alla funzione mistero precedentemente descritta

```
A = input('Inserisci un numero intero positivo: ');
ris = mistero(2:A, A);
disp(ris);
```

2. Implementare un frammento di programma in linguaggio Matlab che legga da tastiera un numero intero positivo A. Quindi, chiami la funzione mistero passandole, come primo argomento, un vettore contenente tutti i numeri interi compresi fra 2 e A e, come secondo argomento, il numero A.
   Infine, stampi a video il valore ritornato dalla chiamata alla funzione mistero precedentemente descritta
3. Spiegare in maniera sintetica quale problema risolve il frammento di programma implementato al punto 2.

La funzione mantiene nel vettore v tutti i numeri che non sono strettamente divisibili per un intero minore o uguale ad n e maggiore di 1

TDE 25 Gennaio 2010

Un supermercato ha memorizzato il proprio archivio di scontrini nel file scontrini.mat, che contiene l'array struttura scontrini i cui elementi hanno i seguenti campi:
IDcliente: id numerico del cliente (>0)
totale: totale della spesa in Euro
punti: punti premio extra associati alle promozioni

Per ogni spesa, viene assegnato al cliente un quantitativo di punti premio pari alla somma dei punti più un ulteriore punto premio per ogni 10 euro spesi.
Scrivere uno script in MATLAB che legge il file scontrini.mat e costruisce un opportuno array struttura saldo, contenente per ciascun cliente, l'ID del cliente e il totale dei suoi punti premio.

Nota: Si faccia attenzione al fatto che un cliente può comparire in più di uno scontrino

Funzioni Variabile

Function Handles

Variabili funzioni

Matlab permette di assegnare a variabili valori di tipo “funzione”

Un valore di tipo funzione può essere assegnato a una variabile (quindi passarlo come parametro), detta handle

l’hanlde è una variabile: quindi può essere utilizzato per trasmettere una funzione ad un’altra funzione

l’hanlde è una funzione: quindi all’handle possono essere passati alcuni argomenti per valutare la funzione

Assegnamento di un valore di tipo funzione

Handle di una funzione esistente

f = @nome_funzione

- Esempio

```
>> seno=@sin
seno = @sin
>> seno(pi/2)
ans = 1
```

f = @(x,y…)<expr>

Handle di una funzione definita ex-novo

- x,y,… sono i parametri della funzione
- <expr> è un’espressione che calcola il valore della funzione
- Esempio

```
>> sq=@(x)x^2
sq = @(x)x^2
>> sq(8)
ans =  64
```

Esempi

```
% definizione della funzione inline
h = @(x) -x
% valutazione della funzione h nei punti 2 e 9
h(2);
h(9);

% g che viene definita in maniera tale da operare su vettori
g = @(x)x.^2
g(2)
g([2 : 2])
g([-2 : 2])

% è possibile "comporre" le funzioni (si da l'output di g come input di h)
h(g([-2 : 2]))
% NB: non è possibile sommare i function hanldes
g + h
```

Funzioni di Ordine Superiore

Funzioni i cui parametri sono function handles

Funzioni di ordine superiore

Se un parametro di una funzione **f** è un handle (cioè contiene un valore di tipo funzione) allora **f** è una **funzione di ordine superiore**

L’handle passato come parametro consente ad **f** di invocare la funzione nell’hanlde

Esempi

```
% creo una funzione per fare quello che fa g
function y = elevaAlQuadrato(x)
y = x.^2;

elevaAlQuadrato(8)

% faccio una funzione di ordine superiore
per valutare se una funzione (passata come argomento mediante un
function handle ) è idempotente in un punto x
% se f(f(x)) == f(x)
function res = controllaSeIdempotente(f , x)
if ( f(f(x)) == f(x))
    res = 1;
else
    res = 0;
end
```

Esempi

```
% chiamata della funzione di ordine superiore
controllaSeIdempotente(g , 1)

% ma non funziona se passo la funzione elevaAlQuadrato
controllaSeIdempotente(elevaAlQuadrato , 1)
Error using elevaAlQuadrato (line 2)
Not enough input arguments.

% occorre creare un function handle per elevaAlQuadrato
controllaSeIdempotente(@elevaAlQuadrato, 1)
% oppure
controllaSeIdempotente(@(x)elevaAlQuadrato(x) , 2)

% il nome delle variabili utilizzate per definire function handle non
conta
k=@(asd) elevaAlQuadrato(asd)
controllaSeIdempotente(k , 2)
```

Funzioni di ordine superiore

Esempio: funzione map che applica una funzione f a tutti gli elementi conetenuti nel parametro vin e ritorna i risultati in vout

```
function [vout]=map(f, vin)
    for ii=1:length(vin)
        vout(ii)=f(vin(ii));
    end;
```

handle

Invoca la funzione passata come argomento

```
>> A=[1,2,3,4,5,6];
>> map(sq,A)
ans = 1  4  9  16  25  36
```

A cosa serve la funzione map?

Non tutte le funzioni possono essere applicate a vettori

Si prenda -- ad esempio– la funzione per vedere se un numero è primo (la si ottiene facilmente a partire dai codici sviluppati nella prima parte del corso)

Come facciamo ad applicare la funzione a tutti gli elementi del vettore

- Scriviamo un ciclo esplicito
- Chiamiamo map(@controllaSePrimo, vett)

Esempio: funzione accumulatore

Funzione accumulatore: `[x] =acc(f, a, u)`

- f è una funzione con due argomenti, con elemento neutro u
- f viene applicata in maniera *cumulativa* a tutti gli elementi di a nel seguente modo:
- applico f ad a(1) e all'elemento neutro, f(u, a(1)),
- Passo al secondo elemento e applico f al risultato dell'operazione precedente e con a(2), i.e, f(f(u,a(1)), a(2)) ..
- Fino a f(…f(f(f(u,a(1)),a(2)),a(3))…,a(length(a)))

```
function [x]=acc(f, a, u)
    x=u;
    for ii=1:length(a)
        x=f(x, a(ii));
    end
```

Esempio: funzione accumulatore

Funzione sommatoria: `function [s]=sommatoria(a)`

- calcola la sommatoria degli elementi di a

```
function [s]=sommatoria(a)
    som=@(x,y)x+y;
    s=acc(som,a,0);
```

- calcola il prodotto degli elementi di a

```
function [s]=produttoria(a)
    prd=@(x,y)x*y;
    s=acc(prd,a,1);
```

- Sono le alternative a sum e prod (built in) e alle implementazioni con cicli tipo

```
p = 1;
for ii = 1 : numel(x)
   p= p * x(ii);
end
```

Questa è la variabile che funziona da accumulatore… inizializzata all'elemento neutro

Altri Esercizi

Esempio

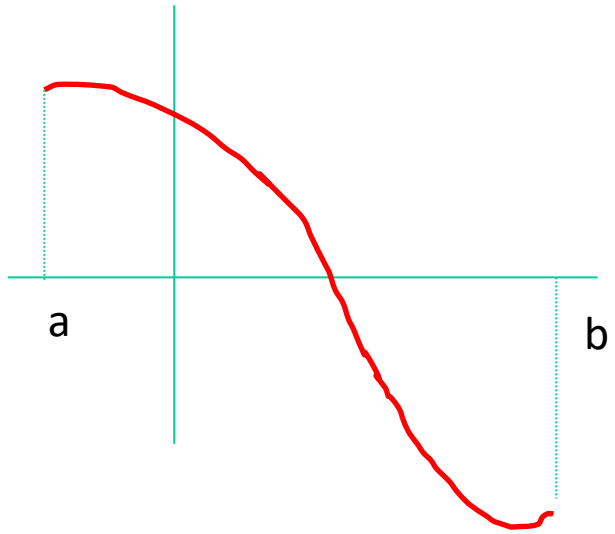
Scrivere una funzione ricorsiva *palindroma* che controlla se una stringa è palindroma

Esempio dal 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 ad f, gli estremi dell'intervallo e determina se la funzione ha uno zero nell'intervallo

TODO:

Utilizzando il metodo di bisezione e il teorema di Bolzano, scrivere una funzione **ricorsiva 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.