LABORATORIO DI PROGRAMMAZIONE 2
Corso di laurea in matematica

Algoritmi ricorsivi

Marco Lapegna
Dipartimento di Matematica e Applicazioni
Universita' degli Studi di Napoli Federico II

wpage.unina.it/lapegna

Partiamo da un esempio

Somma dei primi N numeri interi

$$S = 1 + 2 + \ldots N = \sum_{k=1}^{N} k$$

Un possibile algoritmo ITERATIVO

```
procedure somma (in N; out S)
var  N, S:  integer
S = 0
 for k = 1 to N
     S = S + k
 endfor
end
```

S = 0
S = 0 +1 = 1
S = 1 + 2 = 3
S = 3 + 3 = 6
S = 6 + 4 = 10

IDEA: sommare un numero alla somma parziale dei precedenti

Algoritmi iterativi

l’idea incrementale (o passo-passo) puo’ essere sintetizzata come la risoluzione di una sequenza di istanze dello stesso problema, che a partire dall’istanza piu’ semplice, calcola la soluzione aggiornando la precedente (ad ogni passo e’ disponibile un risultato parziale)

La risoluzione di una istanza viene effettuata attraverso una stessa operazione che coinvolge ad ogni passo la soluzione dell’istanza precedente

l’applicazione dell’approccio incrementale alla soluzione delle formule ricorrenti e’ detta approccio iterativo e il conseguente algoritmo e’ un algoritmo iterativo

un algoritmo iterativo e’ generalmente basato su strutture di iterazione (for, while, repeat)

Approccio alternativo: divide et impera

- L'idea di fondo dell'approccio ***divide et impera*** e' quella di suddividere il problema in due o piu' sottoproblemi (risolvibili piu' semplicemente), applicando ancora a tali sottoproblemi la stessa tecnica suddivisione.

- Un algoritmo basato sul *divide et impera* genera una sequenza di istanze piu' semplici del problema, fino all'istanza che non e' ulteriormente divisibile, e che ha **soluzione banale**

- *In genere la soluzione del problema di partenza deve poi essere ricostruita a partire dalla soluzione banale*

Somma primi N numeri

"somma dei primi N numeri naturali" si puo' riformulare come

$$S = N + \sum_{k=1}^{N-1} k$$

Cioe' N + "somma dei primi (N-1) numeri naturali"

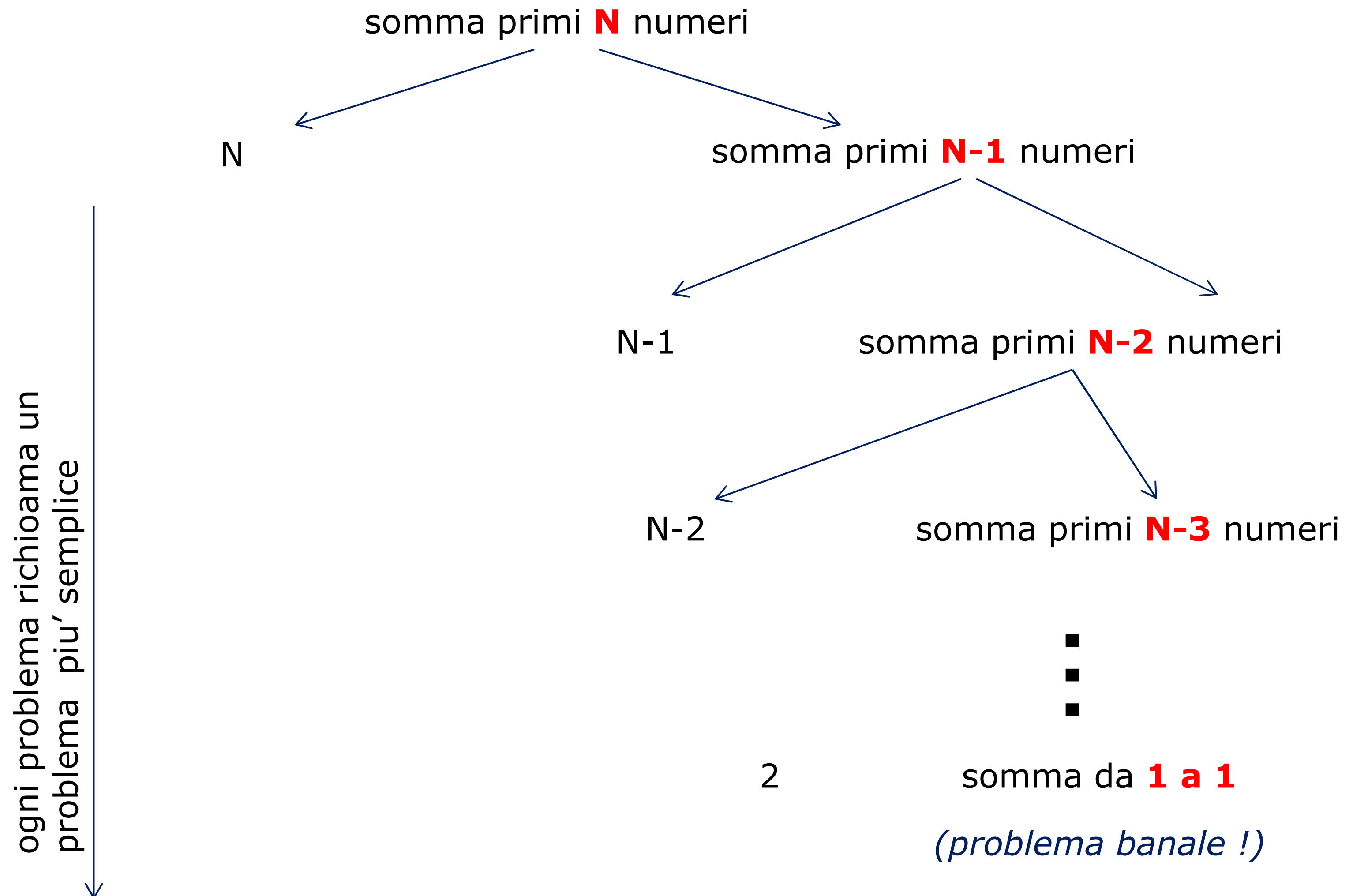
Il calcolo della somma dei primi (N-1) numeri naturali e' una istanza piu' semplice del problema di partenza!

"somma dei primi N-1 numeri naturali" si puo' riformulare come

$$\sum_{k=1}^{N-1} k = (N-1) + \sum_{k=1}^{N-2} k$$

… e cosi' via, fino al problema banale $\sum_{k=1}^{1} k = 1$

somma primi N numeri



Algoritmi ricorsivi

- L' **approccio ricorsivo** e' un modo, alternativo all'*iterazione*, per denotare la *ripetizione di una azione*

- Dal punto di vista del linguaggio, l'approccio ricorsivo si realizza mediante una *function* (o una *procedure*) che al suo interno contiene una chiamata alla *function* stessa; tale processo e' detto *chiamata ricorsiva* (o *autoattivazione*)

- Il calcolo della soluzione di un problema mediante l'approccio ricorsivo, si ottiene distinguendo
    - il caso banale è la cond. iniziale (S = 1 nell'esempio)
    - la chiamata ricorsiva che riproduce la struttura della formula (*N + soluzione del problema della somma dei primi N-1 numeri naturali,* nell'esempio)

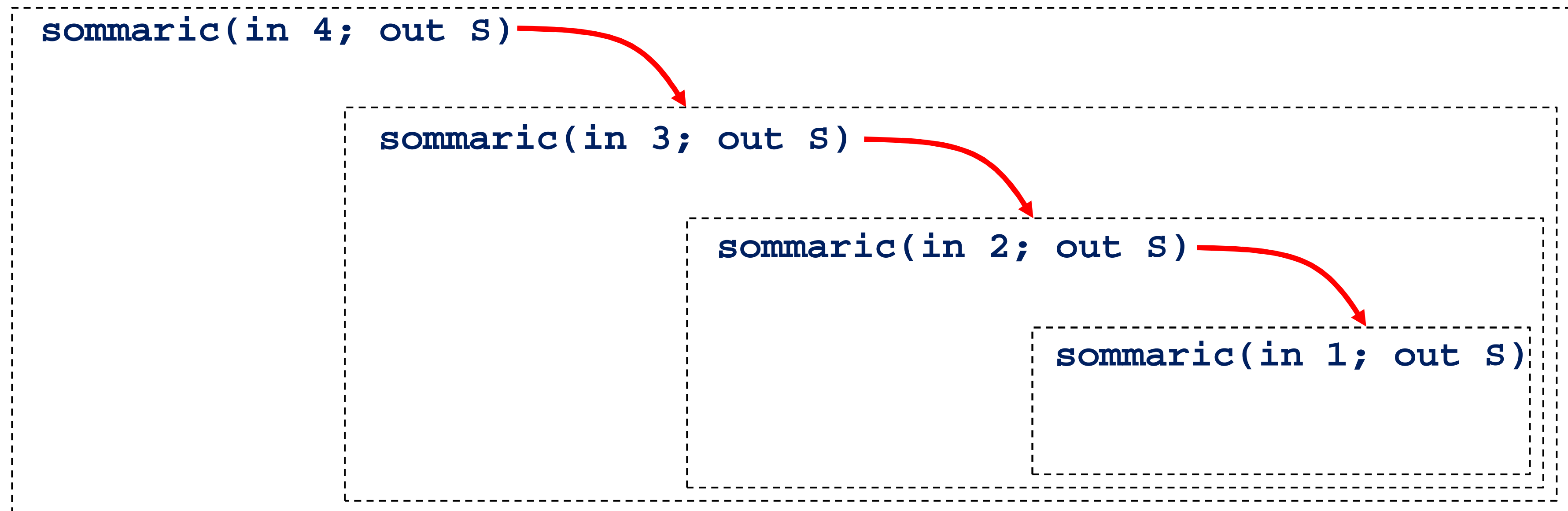
Non viene calcolato nulla finche' non si e' giunti al caso banale

Somma di N numeri (alg. ricorsivo)

```
procedure sommaric (in N; out S)
var N, S: integer
 if (N == 1) then
    /*soluzione problema banale*/
     S = 1
  else
   /*autoattivazione*/
     sommaric ( in: N-1; out: S)
     S = N + S
  endif
end
```

Esecuzione passo passo (N=4)

I fase: autoattivazioni delle funzioni



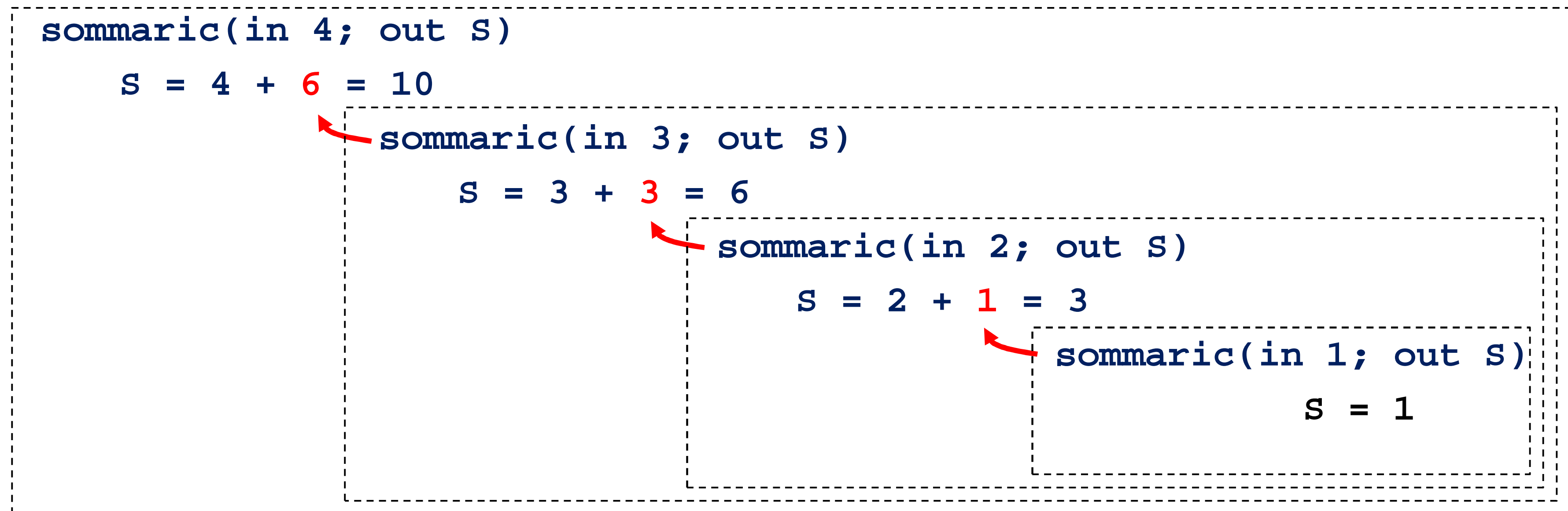
A questo punto non è stato calcolato nessun risultato !!

Sono state effettuate 4 chiamate alla function (1 attivazione e 3 autoattivazioni) ma nessuna attivazione ha portato a termine l'esecuzione !

Siamo arrivati pero' al problema banale`

Esecuzione passo passo (N=4)

II fase: calcolo della soluzione a partire da quella banale



Al termine della prima chiamata (sommaric(in 4; out S)), otteniamo S=10

Iterazione vs ricorsione

RICORSIONE

```
procedure sommaric (in N; out S)
var N, S: integer
 if (N == 1) then
   /*problema banale*/
    S = 1
 else
   /*autoattivazione*/
    sommaric ( in: N-1; out: S)
    S = N + S
 endif
end
```

ITERAZIONE

```
procedure somma (in N; out S)
var  N, S:  integer
S = 0
 for k = 1 to N
     S = S + k
 endfor
end
```

gestione della ricorsione

IN GENERALE

Ogni volta che viene invocata una funzione

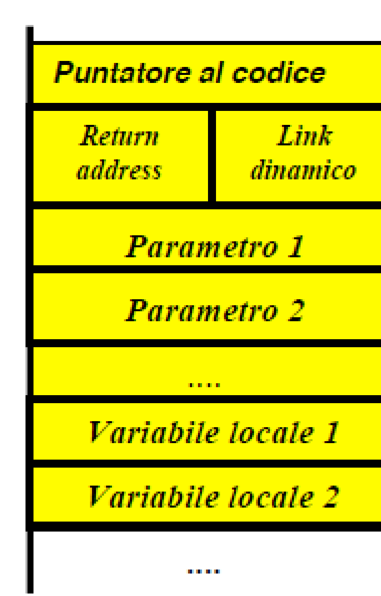
- si crea una nuova istanza
- viene allocata memoria per i parametri e le variabili locali
- si effettua il passaggio dei parametri
- si trasferisce il controllo alla funzione
- si esegue il codice

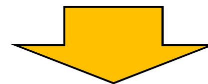
Al momento dell’attivazione viene creata una struttura dati che contiene tutti i dati utili all’esecuzione della funzione (parametri e variabili locali) detta record di attivazione

il record di attivazione

il record di attivazione contiene

- parametri formali
- variabili locali (dichiarate all'interno della funzione)
- indirizzo di ritorno
- indirizzo del codice della funzione





- La dimensione non e' fissa (dipende dalla funzione)
- ad ogni chiamata viene creato un nuovo record
- permane per tutto il tempo di esecuzione della funzione
- e' distrutto al termine della funzione

record di attivazione

Funzioni che richiamano altre funzioni danno luogo a una sequenza di record di attivazioni

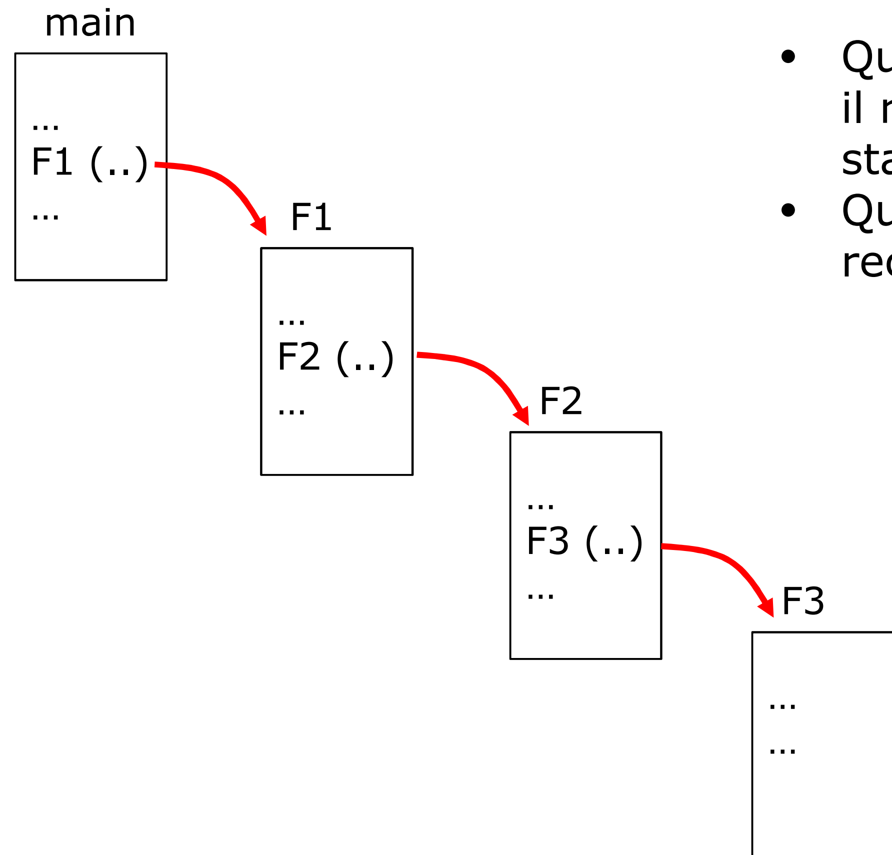
in caso di funzioni ricorsive si genera una sequenza di record della stessa dimensione contenenti dati locali diversi

L'area di memoria in cui sono allocati i record viene gestita come una PILA (struttura LIFO), dove

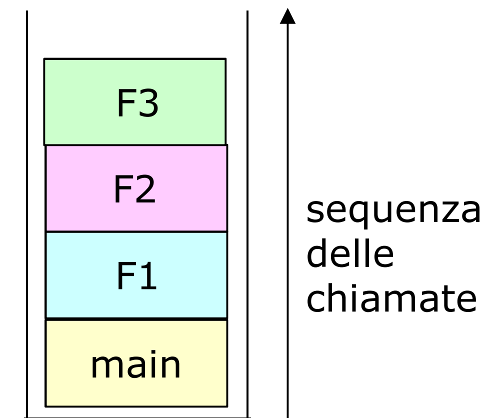
- il main si trova nel fondo della pila
- l'ultima funzione chiamata e' al top della pila

esempio

sequenza delle chiamate



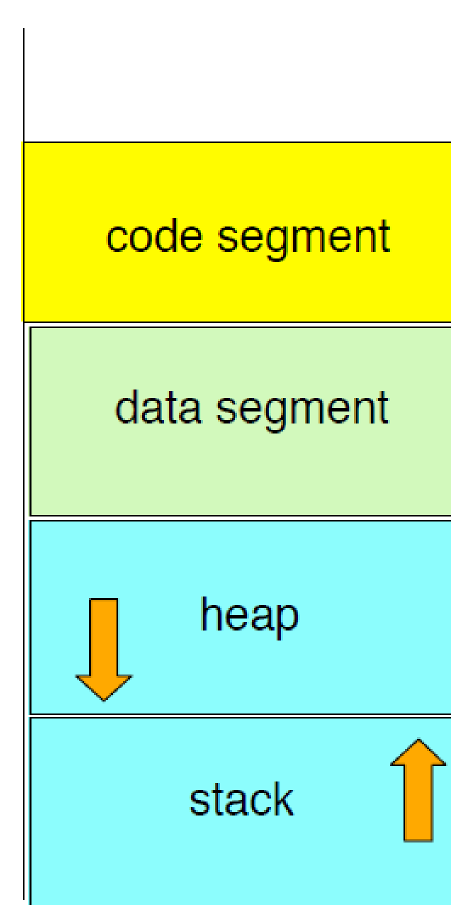
- Quando il main invoca F1 si inserisce il record di attivazione di F1 nello stack
- Quando F1 termina si rimuove il record di attivazione di F1 dallo stack



spazio di indirizzamento

la memoria allocata ad ogni programma in esecuzione (processo) e' divisa in vari segmenti

- code (contiene il codice eseguibile)
- data (contiene le variabili globali, visibili a tutte le funzioni)
- heap (contiene le variabili dinamiche (ad es. allocate con calloc)
- stack (contiene i record di attivazione)



code e data hanno dimensione fissata staticamente al momento della compilazione

stack+heap e' fissa, ma quando lo stack cresce, diminuisce l'area a disposizione dell'heap e viceversa

Massimo Comun Divisore

metodo di Euclide A e B non entrambi nulli (per semplicita' A>B)

Caso banale: B = 0 allora mcd(A,B) = A

Caso generale: A e B non entrambi nulli. Allora A = B*Q + R
e mcd(A,B) = mcd(B,R)

APPROCCIO ITERATIVO

```
procedure mcd(in: A, B; out: M)
var: A. B, Q, R, M integer
while (B /= 0)
   Q = A/B
   R = A – B*Q
   A = B
   B = R
endwhile
M = A
end
```

Massimo Comun Divisore

metodo di Euclide   A e B non entrambi nulli (per semplicita' A>B)

Caso banale: B = 0   allora mcd(A,B) = A

Caso generale: A e B non entrambi nulli. Allora A = B*Q + R
e   mcd(A,B) = mcd(B,R)

APPROCCIO ITERATIVO

```
procedure mcd(in: A, B; out: M)
var: A. B, Q, R, M integer
while (B /= 0)
   Q = A/B
   R = A – B*Q
   A = B
   B = R
endwhile
M = A
end
```

APPROCCIO RICORSIVO

```
procedure mcdric(in: A, B; out: M)
var: A, B, Q, R, M integer
if (B == 0) then
           M = A
else
            Q = A/B
            R = A – B*Q
            mcdric(in: B, R; out M)
endif
end
```

stampa di una lista

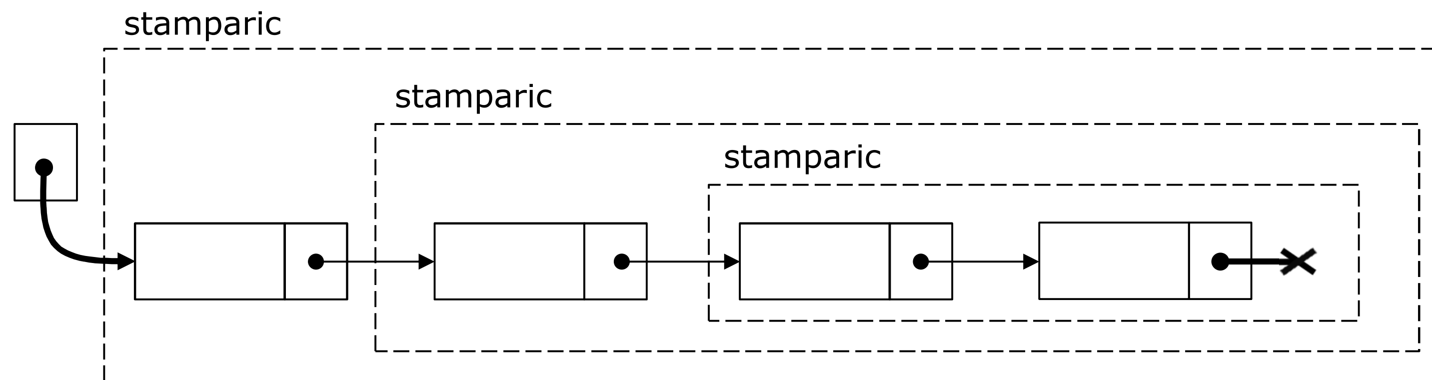
Scrivere una **function** ricorsiva **stamparic** che stampa gli elementi di una lista

Idea

caso banale lista vuota (non fa niente e ritorna)

caso generale (lista non vuota)

stampa il contenuto del campo informazione e stampa lista che parte dal campo puntatore



stampa di una lista

```
procedure stamparic (in: head_list)
var  head_list pointer a list
       if (head_list != NULL) then
              print head_list.info
              stamparic(in: head_list.pointer)
       endif
end
```

OPPURE

```
procedure stamparic (in: head_list)
var  head_list pointer a list
       if (head_list != NULL) then
              stamparic(in: head_list.pointer)
              print head_list.info
endif
end
```

che differenza c'e'

Esercizio: verifica ordinamento di un array

Scrivere una **function** ricorsiva **sorted** che restituisce il valore **TRUE** se gli elementi di un array **A** di interi di dimensione **N**, sono in ordine crescente.

Idea

caso banale (N=1) lista ordinata (restituisce true)

caso generale (N > 1)

si verificano solo A(N-1) e A(N)

se non sono in ordine l'array non e' ordinato (restituisce false)

se sono in ordine bisogna verificare l'array A da 1 a N-1

Esercizio: verifica ordinamento di un array

```
procedure sorted (in: n, A; out R)
var N, A(N) integer
var R logical
    if (N == 1) then
            R = TRUE
    else
         if ( A(n-1)> A(n) ) then
              R = FALSE
         else
              sorted (in: n-1, A, out: R)
         endif
    endif
end
```

Esercizio: ricerca sequenziale

Scrivere una **function** ricorsiva **ric_seq_ricorsiva** che restituisce il valore della posizione **pos** di una **chiave** in un array **A** di dimensione **N** .
L'assenza della chiave è indicata dal valore **pos= 0**

Idea

caso banale (lista vuota) (restituisce pos = 0)

caso generale (lista non vuota)

si verifica solo A(N)

se e' l'elemento cercato si restituisce pos = N

se non e' l'elemento cercato bisogna verificare l'array A da 1 a N-1

Esercizio: ricerca sequenziale

```
procedure ric_seq_ricorsiva (in: N , A, chiave; out: pos)
var: N, A(N), pos integer
   if (N == 0) then
     pos = 0
   else
       if ( A(N) == chiave ) then
         pos = N
       else
         ric_seq_ricorsiva (in: N-1, A, chiave; out: pos)
       endif
   endif
end
```

Esercizio: ricerca binaria

Scrivere una **function** ricorsiva **ric_bin_ricorsiva** che restituisce il valore della posizione **pos** di una **chiave** in un array **A** di dimensione **N** .
L’assenza della chiave è indicata dal valore **pos= 0**

Idea

caso banale (lista vuota) (restituisce pos = 0)

caso generale (lista non vuota)

si verifica solo l’elemento centrale

se e’ l’elemento cercato si restituisce la posizione

se non e’ l’elemento cercato bisogna riapplicare la ricerca nella meta’ superiore oppure in quella inferiore

Esercizio: ricerca binaria

```
procedure  RB_ric(in: vet,  first, last, el; out: pos)
var  vet(),  first, last, med, el: integer

if (first > last)
   pos = 0
else
   med = (first + last) / 2
   if (el == vet(med)) then
        pos = med
   else
       if (el > vet(med))
          RB_bin(in: vet, N, med+1, last, el; out: pos)
       else
          RB_bin(in: vet, N, first, med-1, el; out: pos);
       endif
    endif
endif
```

riepilogo

- Algoritmo ricorsivo: algoritmo la cui soluzione è trovata in termini di soluzioni di versioni "più semplici" dello stesso problema

- Un problema che si risolve "con un ciclo" si può risolvere con la ricorsione(richiamo innestato di un set di istruzioni, proprio come in un ciclo)

- Quello che cambia è che prima di terminare tale sequenza ricorsiva, la sequenza stessa si interrompe, e ne inizia una nuova (nuova *istanza* della stessa sequenza, con i valori dei *parametri piu' piccoli'*) fino a trovare un problema banale