UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Dipartimento di Ingegneria
"Enzo Ferrari"

Algoritmi e Strutture Dati

Ricorsione lineare

RICORSIONE LINEARE

AGENDA

- ✓ Definizione di ricorsione
- ✓ Record di Attivazione e Ricorsione
- ✓ Ricorsione tail
- ✓ Trasformazione ricorsione tail e iterazione

LA RICORSIONE

Definizione

Una funzione matematica è definita per ricorsione (o per induzione) quando è espressa in termini di se stessa

il fattoriale $f(n) = \begin{cases} 1 & \text{se } n=0 \\ n*f(n-1) & \text{se } n>0 \end{cases}$

La base teorica è il principio di induzione:

- se una proprietà vale per un certo naturale $n=n_0$
- e si può dimostrare che, assumendola valida per n, essa è valida anche per n+1,

allora la proprietà vale $\forall\ n \geq n_0$.

Una funzione ricorsiva si specifica definendo:

- quanto vale in un caso detto base
- come si può ricondurre il generico caso “di grado n” a uno o più casi più semplici (di grado < n).

RICORSIONE E PROGRAMMAZIONE

- Una funzione di un linguaggio di programmazione è un servitore che realizza l'astrazione di funzione matematica.
- Il servitore, a sua volta, può essere cliente di altri servitori.
- Come caso particolare, un servitore può essere cliente di se stesso (**funzione ricorsiva***)*

ESEMPIO 1 : il fattoriale

```
unsigned long fattoriale(unsigned long n);
<se n vale 1, restituisci 1;
    altrimenti,
    calcola il fattoriale di n-1
   restituisci tale valore moltiplicato per n>
```

CODIFICA

```
unsigned long fattoriale(unsigned long n) {
  if (n == 1) return 1;
    else
      return n * fattoriale(n-1); }
```

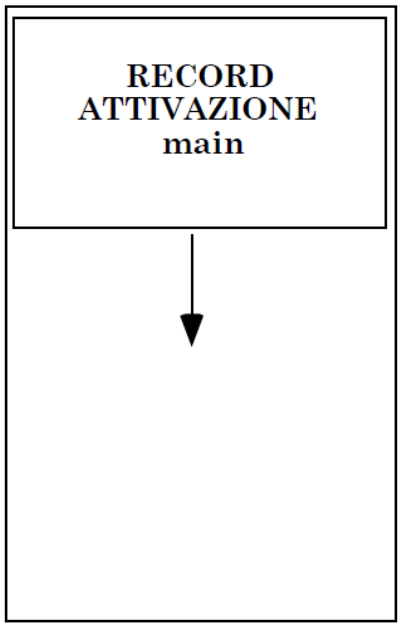
RECORD di ATTIVAZIONE e RICORSIONE

Mostriamo lo stack dei record di attivazione per il seguente programma:

```
unsigned long fattoriale(unsigned long n) {
  if (n == 1) return 1;
    else
      return n * fattoriale(n-1); }

void main(){
  int x = 2, y;
  y = fattoriale(x);
}
```

All'inizio dell'esecuzione:



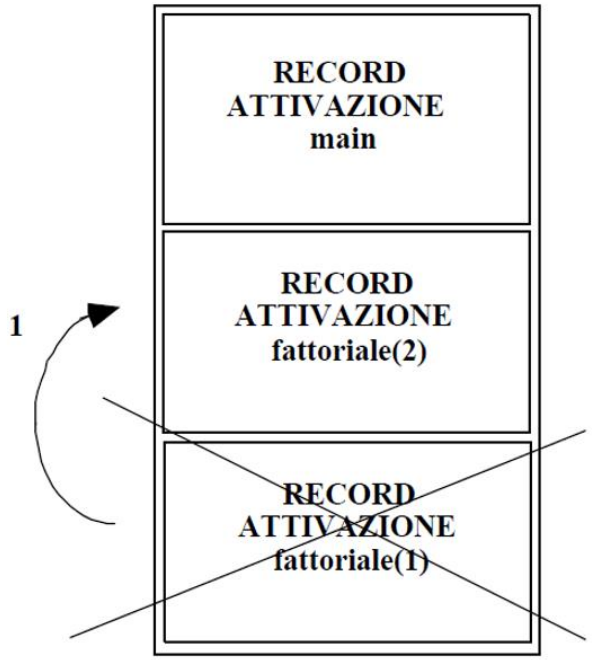
RECORD di ATTIVAZIONE e RICORSIONE (2)

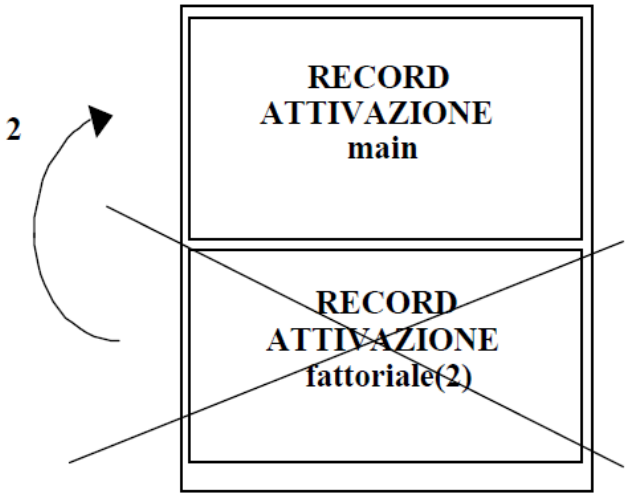
Dopo la prima attivazione (fattoriale(2)):

Dopo la seconda attivazione (fattoriale(1)):

RECORD di ATTIVAZIONE e RICORSIONE (3)

Termine della seconda attivazione (return):





Termine della prima attivazione (viene restituito al main il valore 2 e questo stampa il risultato):

ESEMPIO 2

Calcolare la somma dei primi N numeri positivi

SPECIFICA ITERATIVA

- ripeti per N volte l'operazione elementare

```
sum = sum + i
```

SPECIFICA RICORSIVA

- considera la *somma dei primi N numeri positivi* **(1+2+3+...+(N-1)+N)** come la somma di due termini: **(1+2+3+...+(N-1))** + **N**
- il primo addendo è *la somma dei primi N-1 numeri positivi*
- il secondo addendo è un valore singolo
- è facile identificare un caso base: la *somma del primo numero positivo* vale 1.

```
int sommaFinoA(int n){
  if (n == 1) return 1; /* caso base */
    else
      return sommaFinoA(n-1) + n; /* ricorsione */
}
```

ESEMPIO 3

Calcolare il minimo di una sequenza di elementi $[a_1, a_2, a_3, \ldots a_n]$

SPECIFICA RICORSIVA

- ✓ Il minimo della sequenza $[a_1, a_2, a_3, \ldots a_n]$ è il minimo tra
  - a_1
  - il minimo della sequenza $[a_2, a_3, \ldots a_n]$

- ✓ il minimo della sequenza [a] è a (caso base)

L'algoritmo per trovare il minimo si esprime allora così :

$\min([a_1, a_2, a_3, \ldots a_n]) = \min(a_1, \min([a_2, a_3, \ldots a_n]))$

$\ldots. = \min(a_1, \min([a_2, \min([a_3,\ldots])])])$

Si inizia a sintetizzare il risultato solo quando si sono aperte tutte le chiamate ovvero quando si applica $\min([a_n]) = a_n$.

ESEMPIO 4

Calcolare la lunghezza di una sequenza di elementi [a_1, a_2, a_3, ... a_n]

SPECIFICA RICORSIVA

✓ Data la sequenza S

$$lung(S) = \begin{cases} 0 & \text{se } S = [\,] \\ 1 + lung(S1) & \text{se } S = [\,a, S1\,] \end{cases}$$

Ad esempio

lung([a, 3, 56, h, #]) = (1 + lung([3, 56, h, #])) = (1 + (1 + lung([56, h, #])))
= (1 + (1 + (1 + lung([h, #])))) = (1 + (1 + (1 + (1 + lung([#])))))
= (1 + (1 + (1 + (1 + (1 + lung([ ])))))) = (1 + (1 + (1 + (1 + (1 + 0)))))
..... = (1 + 4) = 5

Anche in quest'esempio si inizia a sintetizzare il risultato solo quando si sono aperte tutte le chiamate.

ESEMPIO 5

Verificare l'appartenenza di un elemento b ad una sequenza di elementi $[a_1, a_2, a_3, \ldots a_n]$

SPECIFICA RICORSIVA

- b non appartiene alla sequenza vuota [] (caso base)
- b appartiene alla sequenza $[a_1, a_2, a_3, \ldots a_n]$
  se $b = a_1$ oppure b appartiene alla sequenza $[a_2, a_3, \ldots a_n]$

$$in(S, b) = \begin{cases} falso & \text{se } S = [\,] \\ vero & \text{se } S = [b, S1] \\ in(S1, b) & \text{se } S = [a, S1] \end{cases}$$

Ad esempio

in([a, b, 3, $], 3) = in([b, 3, $], 3) = in([3, $], 3) = true = in([a, b, 3], c)

= in([b, 3], c) = in([3], c) = in([ ], c) = false

Il risultato viene sintetizzato via via che le chiamate ricorsive si succedono.

ESEMPIO 6

Calcolare il Massimo Comune Divisore fra due numeri (Algoritmo di Euclide)

SPECIFICA RICORSIVA

$$MCD(m, n) = \begin{cases} m & \text{se } m=n \\ MCD(m-n, n) & \text{se } m>n \\ MCD(m, n-m) & \text{se } m<n \end{cases}$$

Ad esempio

mcd(36, 15) = mcd(21, 15) = mcd(6, 15) = mcd(6, 9) = mcd(6, 3)
= mcd(3, 3) = 3

```
int mcd(int m, int n){
  if (m == n) return m;
    else
      if (m > n) return mcd(m-n, n);
        else return mcd(m, n-m);}
```

Il risultato viene sintetizzato via via che le chiamate ricorsive si succedono.

RICORSIONE E RICORSIONE TAIL

Definizione di tail recursion

*Si ha **tail recursion** quando la chiamata ricorsiva è l'ultima istruzione eseguita dalla funzione prima di terminare*

Ovviamente, solo la ricorsione lineare può essere tail.
Riconsideriamo gli esempi precedenti:

- ESEMPIO 1 (fattoriale): non è tail-recursive, perché dopo la chiamata ricorsiva occorre applicare la moltiplicazione:

  n * fattoriale(n-1)

- ESEMPIO 2 (somma fino a N): non è tail-recursive, perché dopo la chiamata ricorsiva occorre fare la somma:

  n + sommaFinoA(n-1)

- ESEMPIO 6 (MCD): è tail-recursive, perché la chiamata ricorsiva è l'ultima istruzione eseguita

RICORSIONE E RICORSIONE TAIL (2)

Una funzione ricorsiva non-tail computa “all’indietro”: al passo i-esimo non è disponibile nulla e il risultato viene sintetizzato mentre le chiamate si chiudono.
È quindi necessario conservare lo stato della computazione prima di fare la chiamata ricorsiva, perché servirà al ritorno.

ESEMPIO:
Nell’ ESEMPIO 2 è indispensabile conservare n:

```
sommaFinoA(3) = sommaFinoA (2) + 3 = (sommaFinoA(1) + 2) + 3
```

Una funzione ricorsiva tail computa in avanti, esattamente come un ciclo: al passo i-esimo, è disponibile il risultato parziale i-esimo.
Non è quindi necessario conservare lo stato della computazione.

ESEMPIO:
Nell’ ESEMPIO 4 lo stato della computazione di ogni chiamata(m,n) serve alla chiamata stessa:

```
mcd(36,15) = mcd(21,15) = mcd(6,15) = mcd (6, 9) =
       mcd (6, 3) = mcd (3, 3) = 3
```

ITERAZIONE E RICORSIONE TAIL

La ricorsione tail dà luogo a un processo computazionale di tipo iterativo. Computando “in avanti”, non richiede di conservare lo stato quindi può essere ottimizzata e resa efficiente come un ciclo:

- non c’è bisogno di allocare nuova memoria per i parametri di ogni chiamata in quanto si può (concettualmente) riutilizzare la precedente
- così facendo, l’occupazione di memoria è esattamente quella di un ciclo che faccia la stessa computazione.

MA ATTENZIONE:
di norma i compilatori C non riconoscono (e quindi non ottimizzano) la ricorsione tail

- Il compilatore GCC con l’opzione -O2 compie l’ottimizzazione tail

ma non è detto che sia così per sempre...

- i linguaggi logici e funzionali (Prolog, Lisp) non hanno strutture cicliche, e usano al loro posto proprio la ricorsione tail (e la ottimizzano)

Trasformazioni in RICORSIONE TAIL

Ricorsione "non-tail" in ricorsione tail

- la ricorsione non-tail computa all'indietro
- la ricorsione tail computa in avanti

```
f(pForm) {
    <corpo della fun>
    op(f(pAtt),var)
}
```

```
f(pForm, ris) {
    <corpo della fun>
    f(pAtt, op(var'))
}
```

Occorre riportare il risultato parziale della computazione.

Ricorsione tail in Iterazione

- La condizione if diventa un ciclo while e si toglie la chiamata ricorsiva
- corpo e condizione di if e del ciclo rimangono immutati

```
if (condizione) {
    <corpo funzione>
    <chiamata ricors.>
}
```

```
while (condizione) {
    <corpo del ciclo>
}
```

È spesso necessario aggiungere nuovi parametri nell'intestazione della funzione tail-ricorsiva, per portare avanti" le variabili di stato.

ESEMPIO 1 - bis

Trasformare la soluzione ricorsiva non-tail dell'ESEMPIO 1 in tail

SOLUZIONE RICORSIVA NON-TAIL

```
long fattoriale(long n) {
  if (n == 1) return 1;
   else
    return n * fattoriale(n-1); }
```

SOLUZIONE RICORSIVA TAIL

```
long fattoriale_tail(long n, long p) {
  if (n == 1) return p;
   else
     return fattoriale _tail(n-1, p*n); }
```

CHIAMATA

```
long x = fattoriale_tail(n,1);
```

ESEMPIO 1 - ter

Trasformare la soluzione ricorsiva tail in soluzione iterativa

SOLUZIONE RICORSIVA TAIL (trasformata)

```
long fattoriale_tail(long n, long p, int i) {
  if (i <= n) {
      p *= i; i++;
      return fattoriale_tail(n, p, i);
   } else
      return p;}
long x2 = fattoriale_tail(6, 1, 1);
```

SOLUZIONE ITERATIVA

```
long fattoriale(long n) {
  int i = 1; long p = 1;
  while (i<=n) { p *= i;
                 i++; }
  return p; }
long x1 = fattoriale(6);
```

ESEMPIO 2 - bis

Trasformare la soluzione ricorsiva non-tail dell'ESEMPIO 2 in tail

SOLUZIONE RICORSIVA NON-TAIL

```
int sommaFinoA(int n){
  if (n == 1) return 1;
    else
      return sommaFinoA(n-1) + n;
}
```

SOLUZIONE RICORSIVA TAIL

```
int sommaFinoA_tail(int n, int res){
  if (n == 1) return 1 + res;
    else
      return sommaFinoA_tail(n-1, n + res);
}
```

CHIAMATA

```
int x = sommaFinoA_tail(n,0);
```

ESEMPIO 4 - bis

Trasformare la soluzione ricorsiva tail dell'ESEMPIO 4 (MCD) in iterativa

SOLUZIONE RICORSIVA TAIL RISCRITTA

```
int mcd(int m, int n){
 if (m != n) {
    if (m > n) m = m-n;
     else n = n-m;
    return mcd(m, n);
  } else
    return m;
}
```

SOLUZIONE ITERATIVA

```
int mcd(int m, int n){
     while (m != n)
         if (m > n) m = m-n;
          else n = n-m;
     return m; }
```

Considerazioni finali su ITERAZIONE E RICORSIONE

L'approccio iterativo
richiede di vedere la soluzione del problema "tutta insieme" in termini di mosse elementari (vedi ESEMPIO 2).
Le soluzioni iterative sono generalmente più efficienti di quelle ricorsive, in termini sia di memoria occupata sia di tempo di esecuzione.

L'approccio ricorsivo
richiede invece solo di esprimere il problema in termini dello stesso problema in casi più semplici, più qualche elaborazione elementare.
Le soluzioni ricorsive sono quindi più espressive e molto più compatte di soluzioni iterative.