La Ricorsione

Una funzione matematica è definita ***ricorsivamente*** quando nella sua definizione compare un riferimento a se stessa

La ricorsione consiste nella possibilità di ***definire una funzione in termini di se stessa***

Operativamente, risolvere un problema con un approccio ricorsivo comporta

- **di identificare un "caso base", con soluzione nota**
- **di riuscire a esprimere la soluzione al caso generico *n* in termini dello *stesso problema* *in uno o più casi più semplici*** (n-1, n-2, etc.)

Esempio di ricorsione

Calcolo del fattoriale di un numero

fact(n) = n!

n! vale 1 se n ≤ 0

n! vale n*(n-1)! se n > 0

Codifica:

```
int fact(int n) {
   if (n<=0) return 1;
   else return n*fact(n-1);
}
```

Esempio di esecuzione (1)

```
int fact(int n) {
   if (n<=0) return 1;
   else return n*fact(n-1);
}
```

```
main(){
   int fz,z = 5;
   fz = fact(z-2);
}
```

Si valuta l'espressione che costituisce il parametro attuale (nell'environment del main) e si trasmette alla funzione fact() una copia del valore così ottenuto (3)

Si esegue la chiamata a funzione fact(3)

Esempio di esecuzione (2)

```
int fact(int n) {
   if (n<=0) return 1;
   else return n*fact(n-1);
}
```

```
main(){
   int fz,z = 5;
   fz = fact(z-2);
}
```

fact(3) effettua una nuova chiamata di funzione.
n-1 nell'environment di fact(3) vale 2 quindi viene chiamata fact(2)

Analogamente fact(2) richiama l'esecuzione di fact(1), che a sua volta richiama fact(0)

Esempio di esecuzione (3)

```
int fact(int n) {
   if (n<=0) return 1;
   else return n*fact(n-1);
}
```

```
main(){
   int fz,z = 5;
   fz = fact(z-2);
}
```

La funzione fact(0) lega il parametro n a 0, quindi la condizione (n<=0) è verificata e la funzione fact(0) restituisce 1 e termina la sua esecuzione, restituendo il controllo dell'esecuzione al suo chiamante, cioè fact(1).

Esempio di esecuzione (4)

```
int fact(int n) {
   if (n<=0) return 1;
   else return n*fact(n-1);
}
```

```
main(){
   int fz,z = 5;
   fz = fact(z-2);
}
```

Nella funzione fact(1) l'esecuzione riprende dal calcolo
*n * fact(n-1) = 1* fact(0) = 1 * 1*
quindi restituisce 1 e termina la sua esecuzione, restituendo il controllo dell'esecuzione al suo chiamante, cioè fact(2).

Nella funzione fact(2) l'esecuzione riprende dal calcolo
*n * fact(n-1) = 2* fact(1) = 2 * 1*
… … …

Esempio di esecuzione (5)

```
int fact(int n) {
   if (n<=0) return 1;
   else return n*fact(n-1);
}
```

```
main(){
   int fz,z = 5;
   fz = fact(z-2);
}
```

Nella funzione fact(3) l'esecuzione riprende dal calcolo
*n * fact(n-1) = 3* fact(2) = 3 * 2*
quindi restituisce 6 e termina la sua esecuzione, restituendo il controllo dell'esecuzione al suo chiamante, cioè il main

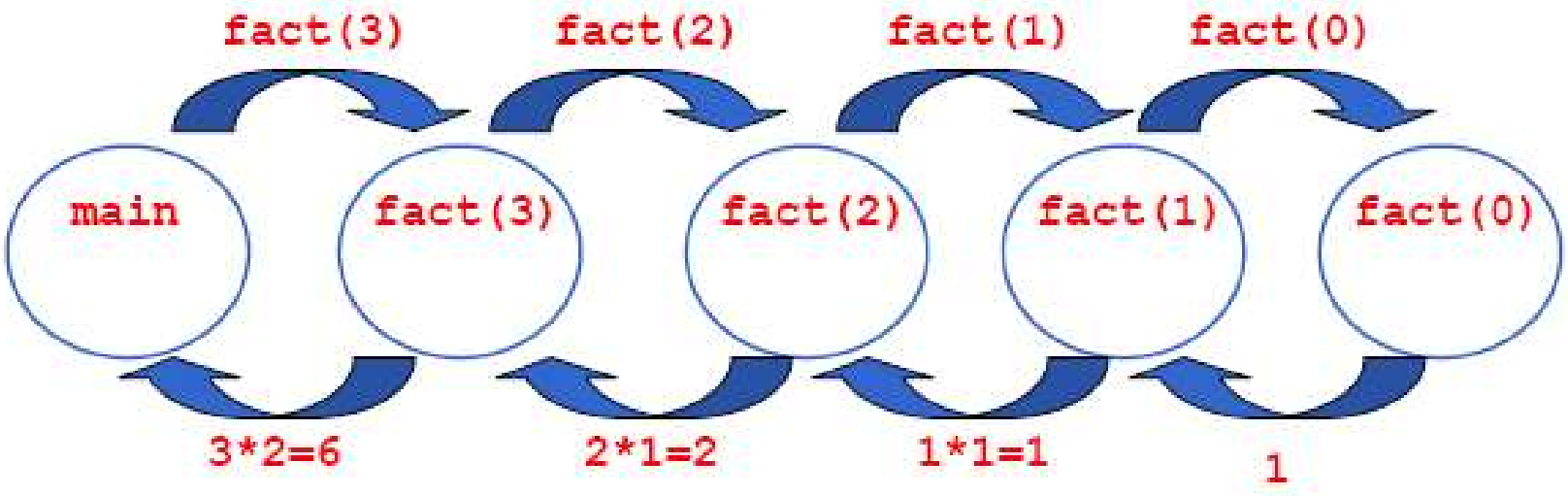
Nel main, il risultato di fact(3) è assegnato alla variabile fz, quindi fz assume il valore 6.

Schema di esecuzione

Il controllo dell'esecuzione segue questo schema:

main → fact(3) → fact(2) → fact(1) → fact(0)

e poi si torna indietro nello stesso percorso a ritroso per restituire il risultato della funzione



Esempio: somma ricorsiva

Problema: calcolare la somma dei primi N interi

Per risolvere il problema con approccio ricorsivo:

1. **identificare un "caso base", con soluzione nota**

   Caso banale: se N=1 ➔ la somma vale 1

2. **esprimere la soluzione al caso generico n in termini dello stesso problema**

   Considerare la somma 1+2+3+...+(N-1)+N come composta di due termini:

   1° termine: somma fino a N-1 (1+2+3+...+(N-1))

   2° termine: il valore N

Esempio: somma ricorsiva (2)

Algoritmo ricorsivo

- Se N vale 1 allora la somma vale 1
- altrimenti la somma vale:
  N + il risultato della somma dei primi N-1 interi

Codifica:

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

Processo computazionale ricorsivo

Negli esempi visti finora si inizia a sintetizzare il risultato

SOLO DOPO che si sono aperte tutte le chiamate, *"a ritroso"*, mentre le chiamate si chiudono

Le chiamate ricorsive decompongono via via il problema, ***ma non calcolano nulla***

Il risultato viene sintetizzato *a partire dalla fine, perché prima occorre arrivare al caso "banale"*:

il caso "banale" fornisce il valore di partenza

poi si sintetizzano, "a ritroso", i successivi risultati parziali

Processo computazionale effettivamente ricorsivo

Processo computazionale iterativo

In questo caso il risultato viene sintetizzato *"in avanti"*
Ogni processo computazionale che computi "in avanti", per accumulo, costituisce una ***ITERAZIONE***, ossia è un *processo computazionale iterativo*

La caratteristica fondamentale di un **processo computazionale *ITERATIVO* è che *a ogni passo è disponibile un risultato parziale***

- dopo k passi, si ha a disposizione il risultato parziale relativo al caso k
- questo ***non è vero* nei processi computazionali *ricorsivi,*** in cui nulla è disponibile fino al caso elementare

Accumulatore del pr. iteratico

Un processo computazionale iterativo si può realizzare anche tramite funzioni ricorsive

Si basa sull'utilizzo di una variabile, detta *accumulatore*, destinata a esprimere *in ogni istante* la soluzione corrente

Si imposta identificando l'operazione di *modifica dell'accumulatore* che lo porta a memorizzare, dal valore relativo al passo k, il valore relativo al passo k+1

Esempio: fattoriale iterattivo (1)

Definizione:

$$n! = 1 * 2 * 3 *\ldots * n$$

Detto $f_k = 1 * 2 * 3 *\ldots * k$:

- $1! = v_1 = 1$
- $(k+1)! = v_{k+1} = (k+1) * v_k$ *per $k \geq 1$*
- $n! = v_n$ *per $k=n$*

Esempio: fattoriale iterattivo (2)

Costruiamo ora una funzione che calcola il fattoriale in modo iterativo

```
int fact(int n){
   int i=1;
   int f=1; /*inizializzazione del fattoriale*/
   while (i <= n)
   { f=f*i;
   i=i+1; }
   return f;
}
```

DIFFERENZA CON LA VERSIONE RICORSIVA: ad ogni passo viene accumulato (nella variabile f) un risultato intermedio

Iterazione e ricorsione TAIL

il corpo del ciclo rimane *immutato*

il ciclo diventa un `if` con, in fondo, la chiamata tail-ricorsiva

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

Tail ricorsiva

```
if (condizione) {
  <corpo del ciclo>
  <chiamata ricorsiva>
}
```

Naturalmente, può essere necessario ***aggiungere nuovi parametri*** nell'intestazione della funzione tail-ricorsiva, per "portare avanti" le variabili di stato

Fattoriale Tail ricorsive

```
int fact(int n){
 return factIt(n,1,1);
}
```

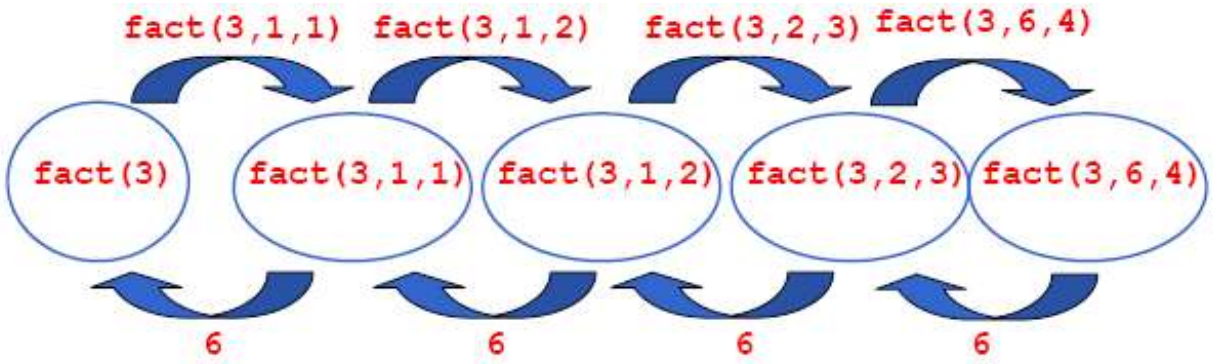
Inizializzazione dell'accumulatore: corrisponde al fattoriale di 1

Contatore del passo

```
int factIt(int n, int F, int i){
   if (i <= n)
      {F = i*F;
      i = i+1;
      return factI(n,F,i);
   }
   return F;
}
```

Accumulatore del risultato parziale

Ricorsione TAIL



ciascuna funzione che effettua una chiamata ricorsiva si sospende, aspetta la terminazione della funzione chiamata e poi termina, cioè

NON EFFETTUA ALTRE OPERAZIONI DOPO

Ricorsione TAIL

La soluzione ricorsiva individuata per il fattoriale è ***sintatticamente ricorsiva*** ma dà luogo a un ***processo computazionale ITERATIVO***

Ricorsione apparente detta **RICORSIONE TAIL**

Il risultato viene sintetizzato *in avanti*

- ogni passo *decompone e calcola*
- *e porta in avanti il nuovo risultato parziale*
- quando le chiamate si chiudono non si fa altro che riportare indietro, fino al chiamante, il risultato ottenuto

Riassumendo …

Una ricorsione che realizza un processo computazionale *ITERATIVO è una ricorsione apparente*

la chiamata ricorsiva è *sempre l'ultima istruzione*

- *i calcoli sono fatti prima*
- *la chiamata serve solo, dopo averli fatti, per proseguire la computazione*

questa forma di ricorsione si chiama *RICORSIONE TAIL* ("ricorsione in coda")