

# RICORSIONE

ricorsione su dati automatici  
(testo Cap. 10)

problemi si dividono in sottoproblemi e

```
int F(.....)
```

```
{  
    double G(....)
```

```
    {  
        .....G(...)  
        ....H(..) ... e così via  
    }
```

```
}
```

e se  $F = G = H$  ?   Ricorsione

```
int F(..)
```

```
{
```

```
....G(..)..
```

```
}
```

```
double G(...)
```

```
{
```

```
....H(..)...
```

```
}
```

stack dei dati

Var locali di F

Var locali di G

Var locali di H

Record d'Attivazione (RA)  
contiene le var locali e anche  
l'indirizzo di ritorno

con ricorsione

```
int F(..)
{
  ....F(..)..
}
```

stack dei dati

Var locali di F

Var locali di F

Var locali di F

una sola funzione, ma tante invocazioni e un RA per le variabili locali di ciascuna invocazione

ci sono calcoli naturalmente ricorsivi :

-il fattoriale di 1 è 1

-il fattoriale di  $n > 1$  è

$$n * (n-1) * (n-2) * \dots 1$$

$$\text{fatt}(n-1)$$


$$\text{fatt}(n)$$

```
int fatt(int n)
```

```
{
```

```
  if(n==1)
```



caso base

```
    return 1;
```

```
  else
```

```
    return n * fatt(n-1);
```

```
}
```

stack dei dati

n=3

n=2

n=1

...fact(3)....

```
int fact(int n)
```

```
{
```

```
  if (n==1) return 1;
```

```
  else
```

```
    return n*fact(n-1);
```

```
}
```



programma  
che esegue

.....

x=fatt(3);

.....

int fatt(int n  
qui

{ if(n<=1) return 1;

else

return n \* fatt(n-1);

}

e  
l'esecuzione  
continua da  
qui

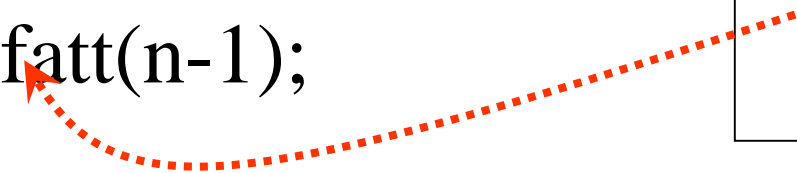
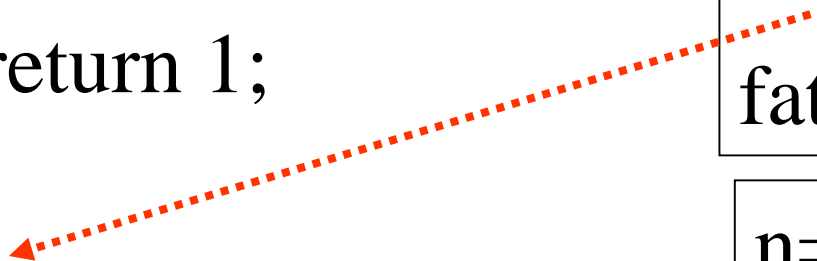
stack dei dati

x= 6

n=3  
fatt(2) 2

n=2  
fatt(1) 1

n=1





# il caso base è importante

```
int fatt(int n)
```

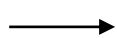
```
{ if(n<=1) return 1;
```

```
else
```

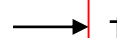
```
    return n * fatt(n-1);
```

```
}
```

fatt(3)



fatt(2)



fatt(1)



fatt(0)



fatt(-1)



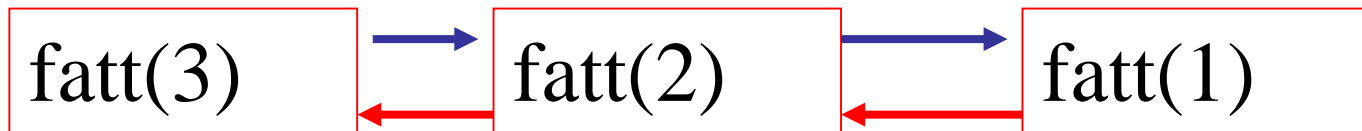
fatt(-2)



all'infinito

in un calcolo ricorsivo

andata



ritorno

ESEMPIO:

determinare se in un array c'è z:

PRE=( $\text{dim} \geq 0$ ,  $A[0..\text{dim}-1]$  è definito)

bool presente(int\* A, int dim, int z)

POST=(restituisce true sse  $A[0..\text{dim}-1]$  contiene z)

caso base:

-se  $\text{dim}==0$  allora l'array è vuoto e quindi la risposta è false

passo induttivo:

-se  $A=[x, \text{resto}]$ , se  $x=z$ , allora true e altrimenti si deve cercare nel resto e questo lo fa l'invocazione ricorsiva:  $\text{presente}(A+1, \text{dim}-1, z)$

```
//PRE=(dim>=0, A[0..dim-1] è definito)
```

```
bool presente(int *A, int dim, int z)
```

```
{
```

```
    if(dim==0)
```

```
        return false;
```

```
    else
```

```
        if(A[0]==z)
```

```
            return true;
```

```
        else
```

```
            return presente(A+1,dim-1, z);
```

```
}
```

```
//POST=(restituisce true sse A[0..dim-1] contiene z)
```

come facciamo per dimostrare  
la correttezza??

come facciamo per le funzioni normali?

```
PRE_f  
int f(...)  
{  
  .....  
  .....g(...)  
  .....  
}  
POST_f
```

vale PRE\_g rispetto ai parametri  
attuali

abbiamo dimostrato che g è  
corretta rispetto a PRE\_g e  
POST\_g

usiamo questo fatto nella prova  
che f è corretta rispetto a PRE\_f  
e POST\_f

allora vale  
POST\_g

con la ricorsione, al posto della prova della correttezza di  $g$ , facciamo la seguente ipotesi induttiva:

assumiamo che l'invocazione induttiva:  
 $\text{presente}(A+1, \text{dim}-1, z)$ ;  
sia corretta, cioè faccia quello che deve

dovremo dimostrare che i parametri attuali soddisfano la PRE

osserva che se

$A[0..\text{dim}-1]$  allora

$$(A+1)[0..\text{dim}-2] = A[1..\text{dim}-1]$$



PRE=( $\text{dim} \geq 0$ ,  $A[0..\text{dim}-1]$  è definito)

presente(  $A$ ,  $\text{dim}$ ,  $z$ )

POST=(restituisce true sse  $A[0..\text{dim}-1]$  contiene  $z$ )



PRE\_ric= ( $\text{dim}-1 \geq 0$ ,  $(A+1)[0..\text{dim}-2]$  è definito)

presente( $A+1$ ,  $\text{dim}-1$ ,  $z$ );

POST\_ric=(restituisce true sse  $(A+1)[0..\text{dim}-2]$   
contiene  $z$ )

prova induttiva (testo 10.2.1):

1) caso base:

PRE < caso base > POST

2) passo induttivo:

- ipotesi induttiva: si assume che le invocazioni ricorsive sono corrette rispetto a PRE e POST

- vale PRE < caso non base > POST

primo caso base:

PRE=( $\text{dim} \geq 0$ , A[0.. $\text{dim}-1$ ] è definita)

if ( $\text{dim} == 0$ ) return false;

POST=(presente restituisce true sse A[0.. $\text{dim}-1$ ]  
contiene z)

## secondo caso base

//PRE=( $\text{dim} \geq 0$ ,  $A[0..\text{dim}-1]$  è definito)

if( $\text{dim} == 0$ )

    return false;

else // ( $\text{dim} > 0$ ,  $A[0..\text{dim}-1]$  definito)

    if( $A[0] == z$ )

        return true;

    else...

//POST=(restituisce true sse  $A[0..\text{dim}-1]$  contiene  $z$ )

## caso induttivo

```
if(A[0]=: (dim>0, A[0..dim-1] è definito) =>  
    PRE_ric= (dim-1>=0, (A+1)[0..dim-2]  
    else   è definito)
```

return presente(A+1,dim-1,z) ;

POST\_ric=(restituisce true sse (A+1)[0..dim-2]=A[1..dim-1] contiene z)

=>

POST=(restituisce true sse A[0..dim-1] contiene z)

è corretto assumere l'ipotesi induttiva?

consideriamo  $\text{dim}=0$ ,  $\text{dim}=1$ ,  $\text{dim}=2, \dots$

la funzione presente è corretta con array vuoto,  
con array con 1 elemento,  
con array con 2 elementi  
e così via ....

quando dimostriamo il caso  $\text{dim}$  usiamo la correttezza del caso  $\text{dim}-1$  che abbiamo dimostrato prima

ma non conta il particolare valore  $\text{dim}$  che consideriamo !!!!

il passo induttivo non dipende da  $\text{dim} \Rightarrow$  vale per ogni  $\text{dim}$

basta fare il passo induttivo 1 sola volta

rivediamo la nostra funzione ricorsiva

```
bool presente(int *A, int dim, int z)
```

```
{  
    if(dim==0)  
        return false;  
    else  
        if(A[0]==z)  
            return true;  
        else  
            return presente(A+1,dim-1,z);  
}
```



possiamo fare lo stesso con un while

```
bool trovato=false;
while(dim>0 && !trovato)
{
    if(A[0]==z)
        trovato=true;
    else
        { A++; dim--; }
}
```

PRE=( $vA=A$ ,  $vdim=dim$ ,  $vA[0..vdim-1]$  definito)

```
bool trovato=false;
while(dim>0 && !trovato)//R
{
    if(A[0]==z)
        trovato=true;
    else
        { A++; dim--; }
}
```

R= ( $trovato \Rightarrow z=vA[vdim-dim]$  e  $dim>0$ ) &&  
( $!trovato \Rightarrow vA[0..vdim-dim-1] \neq z$ )  
&&(0 $\leq dim \leq vdim$ )

questo ciclo è più semplice:

```
bool trovato=false; int i=0;
```

```
while(i<dim && !trovato)
```

```
{ if(A[i]==z)
```

```
    trovato=true;
```

```
i++;
```

```
}
```

$R = (0 \leq i \leq \text{dim}) \&\& (\text{trovato} \text{ sse } z \text{ in } A[0..i-1])$

invariante parla di ciò che è stato fatto

post della ricorsione parla di quello che resta da fare

```
//PRE=(dim>=0, A[0..dim-1] è definito)
```

```
bool presente(int *A, int dim, int z)
```

```
{
```

```
    if(dim==0)
```

```
        return false;
```

```
    else
```

```
        if(A[0]==z)
```

```
            return true;
```

```
        else
```


```
            return presente(A+1,dim-1, z);
```

```
}
```

```
//POST=(restituisce true sse A[0..dim-1] contiene z)
```

presente si può scrivere anche così:

```
bool presente(int *A, int dim, int z)
{
    if(dim==0)
        return false;
    else
        return (A[0]==z) || presente(A+1,dim-1,z);
}
```



scambiare le 2 condizioni significa fare chiamate ricorsive potenzialmente inutili (se  $A[0]==z$ )

faremmo prima l'invocazione ricorsiva e poi il test su  $X[0]$

insomma il test verrebbe fatto “al ritorno” della ricorsione, rischiando di fare invocazioni inutili

vista la valutazione short-cut delle espressioni booleane, il test ci può permettere di terminare il calcolo,

sarebbe come questo

```
bool trovato=false;
while(dim>0)
{
    if(A[0]==z)
        trovato=true;
    A++;
    dim--;
}
```

un ciclo più compatto

```
while(dim>0 && A[0]!=z)
{ A++; dim--; }
```

trovate l'invariante