RICORSIONE

ricorsione su dati automatici

(testo Cap. 10)

problemi si dividono in sottoproblemi e

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

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

}
```

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

e se F = G = H ? Ricorsione

```
inf F(..)
{
  ....G(..)..
}
void G(...)
{
....H(..)...
}
```

stack dei dati
Var locali di F
Var locali di G
Var locali di H

```
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 calcoli ricorsivi :

-il fattoriale di 1 è 1

-il fattoriale di n>1 è

$$\underbrace{n * \underbrace{\boxed{(n-1) * (n-2) * \ldots 1}}_{\text{fatt}(n-1)}}_{\text{fatt}(n)}$$

```
int fatt(int n)
{
 if(n==1)
   return 1;
else
  return n * fatt(n-1);
}
```

caso base

```
...fact(3)....

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

stack dei dati
n=3
n=2
n=1

stack dei dati

programma che esegue

```
……
x=fatt(3);
…..
int fatt(int n)
{ if(n<=1) return 1;
else
  return n * fatt(n-1);
}
```

e l'esecuzione continua da qui

x= 6
n=3 fatt(2 2
n=2 fatt(1) 1
n=1

il caso base è importante

int fatt(int n)

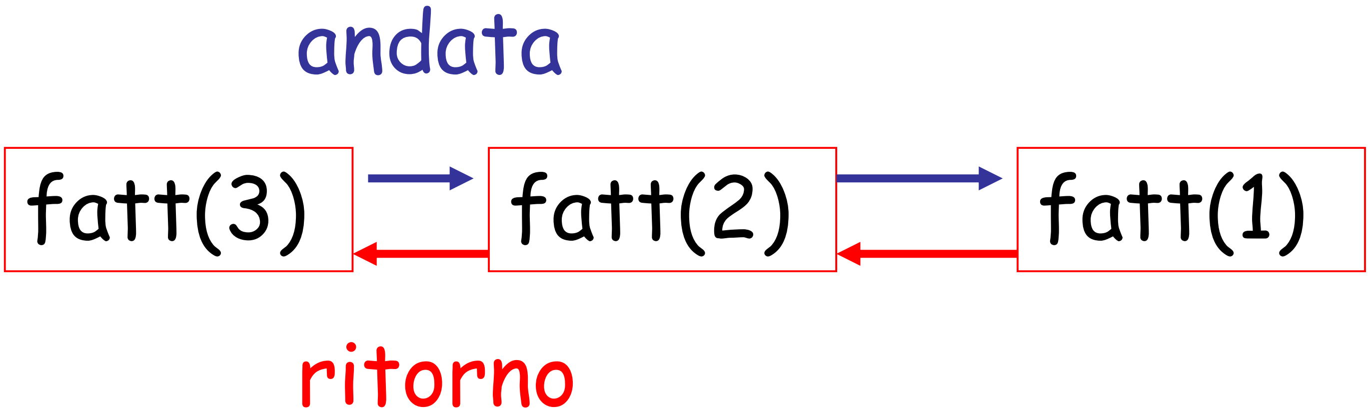
~~{ if(n<=1) return 1;~~

~~else~~

  return n * fatt(n-1);

}

fatt(3) → fatt(2) → fatt(1) → fatt(0) → fatt(-1) → fatt(-2) → all'infinito in un calcolo ricorsivo



ESEMPIO:
determinare se in un array c'è z:

PRE=(dim>=0, A[0..dim-1] def., z def.)

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

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

caso base:

-se dim==0 allora l'array è vuoto e quindi la risposta è false

passo induttivo:

-se A=[x , resto], se x==z, allora true e altrimenti si deve cercare nel resto => presente(A+1,dim-1,z)

PRE

```
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
```

come facciamo per la correttezza??

e per le funzioni normali?

vale PRE_g rispetto ai parametri attuali

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

}
POST_f
```

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

vale POST_g

dobbiamo assumere che
presente(A+1,dim-1,z); sia corretta
rispetto a
PRE_ric= (dim-1>=0, (A+1)[0..dim-2]
def., z def.)
e
POST_ric=(restituisce true sse
(A+1)[0..dim-2] contiene z)

assumere che l'invocazione ricorsiva sia corretta rispetto a PRE e POST significa
assumere che, se prima dell'invocazione vale PRE allora al ritorno vale POST

è quello che si fa sempre con le invocazioni di funzione !
ma in caso di funzione ricorsiva sembra che stiamo mordendoci la coda !!

prova induttiva (testo 10.2.1):

-caso base:
PRE<caso base> POST

-passo induttivo:

- -ipotesi induttiva: le invocazioni ricorsive sono corrette rispetto a PRE e POST
- -vale PRE <caso non base> POST

caso base:

PRE=(dim>=0, A[0..dim-1] def., z def.)

```
if (dim==0) return false;
```

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

secondo caso base

PRE=(dim>=0, A[0..dim-1] def., z def.)

{ if(dim==0)

    return false;

else (dim>0, A[0..dim-1] def., z def.)

    {if(A[0]==z)

    return true;

      else...

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

caso induttivo

if(A[0]==z) then return true

(dim>0, A[0..dim-1] def., z def.)=>
PRE_ric= (dim-1>=0, (A+1)[0..dim-2] def., z def.)

else

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)

ha senso assumere l'ipotesi induttiva?

consideriamo dim=0, dim=1, dim=2,....

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

quando dimostriamo il caso dim
usiamo la correttezza del caso dim-1
che abbiamo dimostrato prima

ma non conta il particolare valore dim che consideriamo !!!!

il passo induttivo riassume tutte le prove in una volta sola 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] def., z def.)

```
bool trovato=false;

while(dim>0 && !trovato)

{ if(A[0]==z)
        trovato=true;

else

        {A++; dim--; }
}
```

R= (trovato=>z=vA[vdim-dim] e dim>0) && (!trovato=> vA[0..vdim-dim-1] !=z) &&(0<=dim<=vdim)

più facile questo ciclo:

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

while(i<dim && !trovato)

{ if(A[i]==z)
        trovato=true;

i++;

}
```

R=(0<=i<=dim)(trovato
sse z in A[0..i-1])

invariante parla del passato

post della ricorsione parla di quello che resta da fare 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 viene fatto "al ritorno" della ricorsione

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

facendolo dopo l'invocazione, rischieremmo di fare invocazioni inutili sarebbe come questo

```
bool trovato=false;

while(dim>0)

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

un ciclo + compatto

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

trovate l'invariante