

RICORSIONE

ricorsione su dati automatici

problemi si dividono in
sottoproblemi e

.... F (.....)

{

..... G (...)

}

e se $G = F$? Ricorsione


ci sono problemi che si prestano ad una soluzione ricorsiva :

-il fattoriale di 0 e 1 è 1

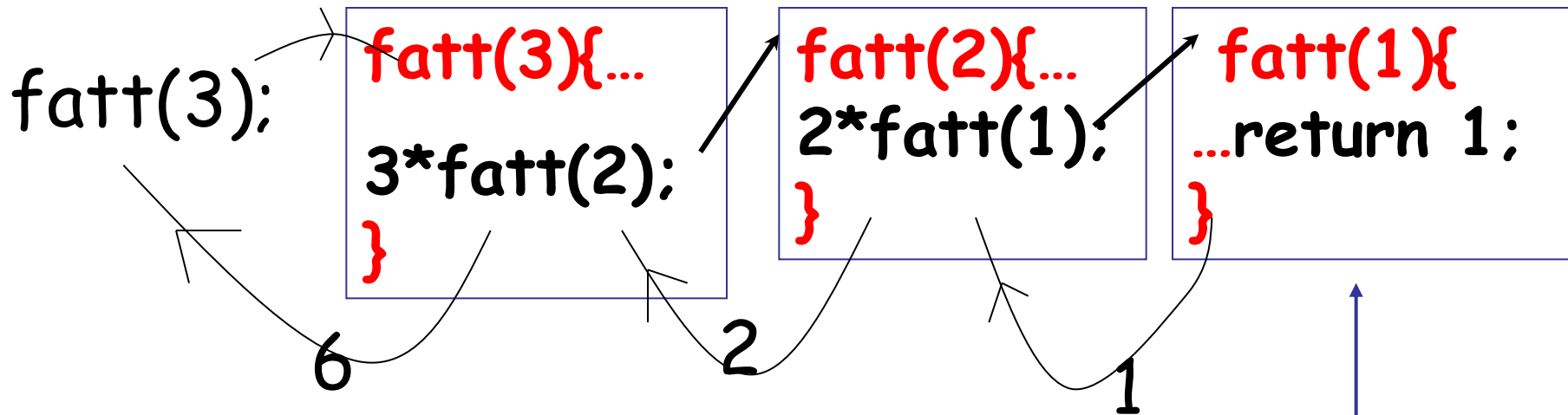
-il fattoriale di $n > 1$ è

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

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


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

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

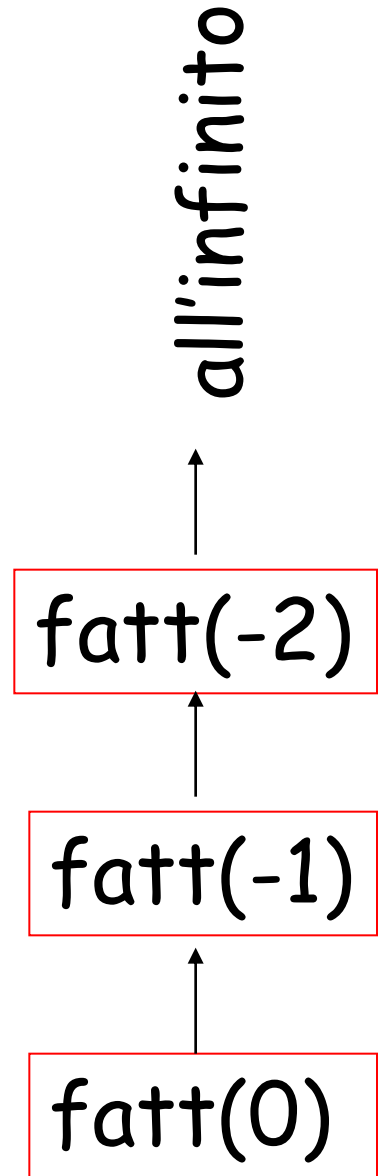
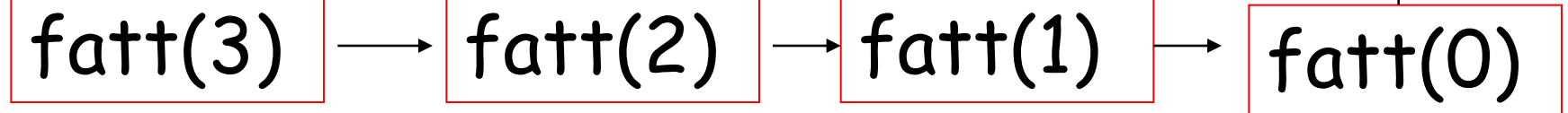


```
int fatt(int n)
{ if(n<=1) return 1;
  else
    return n * fatt(n-1);
}
```

condizione di
terminazione

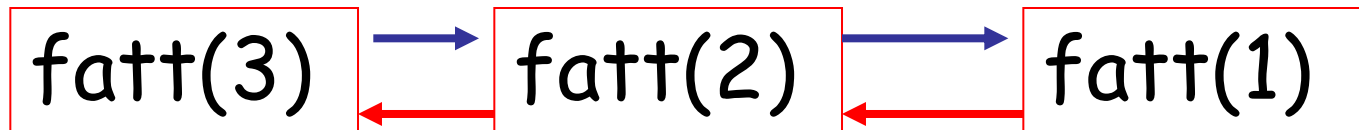
se non avessimo la condizione di
terminazione

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



se tutto va bene:

andata



ritorno

ma che succede
durante questa
esecuzione ?

c'è un solo codice
di `fatt`

e diverse copie
dei dati

programma
che esegue

.....

x=fatt(3);

.....

e l'esecuzione
continua da
qui

int fatt(int n)

{ if(n<=1) return 1;

else

return n * fatt(n-1);

}

stack dei dati

x= 6

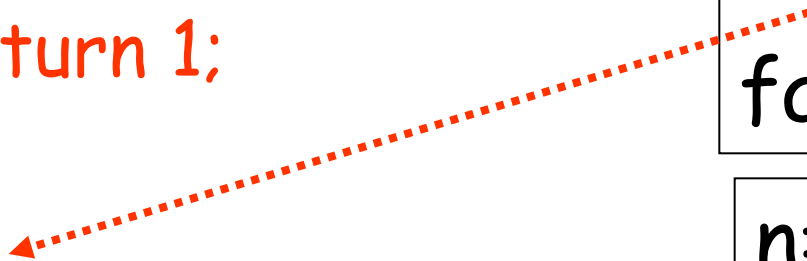
n=3

fatt(2) 2

n=2

fatt(1) 1

n=1



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

caso base:

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

passo induttivo:

-se $A=[x \mid \text{resto}]$, allora se $x==z$, allora true e altrimenti

$\text{presente}(\text{resto}) = \text{presente}(A+1, \text{dim}-1)$

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
```

si assume che l'invocazione ricorsiva
sia corretta rispetto a PRE e POST

cioè che se prima dell'invocazione
vale PRE allora al ritorno vale POST

è quello che si fa sempre con le
invocazioni di funzione !

ma ha senso in caso di invocazione
ricorsiva ?

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=($\text{dim} \geq 0$, $A[0..\text{dim}-1]$ def., z def.)

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

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

passo induttivo:

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

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

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

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

ha senso l'ipotesi induttiva?

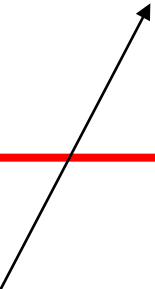
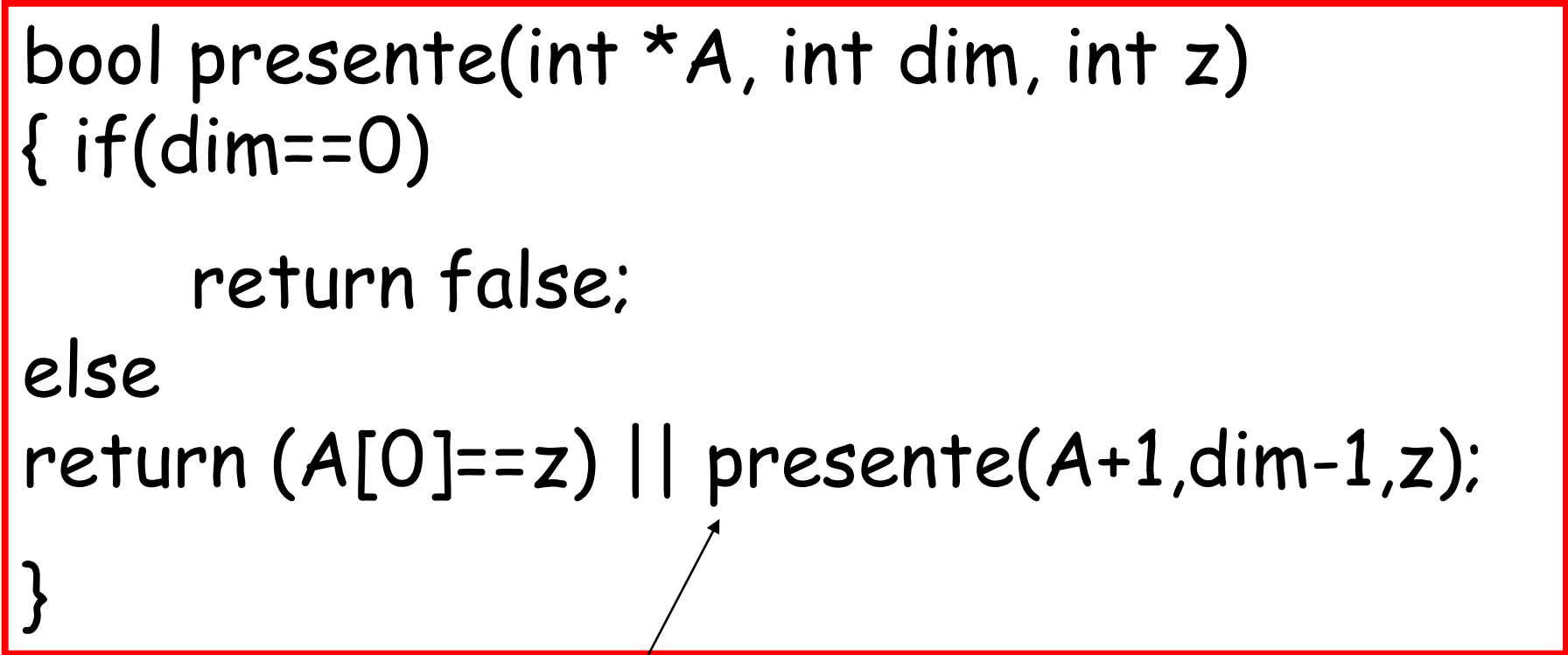
consideriamo $\text{dim}=0$, $\text{dim}=1$, $\text{dim}=2$,

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

passando da dim a $\text{dim}+1$ si richiede la
dimostrazione del passo induttivo!!

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

visto che il test ci può permettere di **interrompere** la ricorsione, facendolo al ritorno, rischiamo di fare invocazioni inutili

equivale più o meno a:

```
while(dim>0)
```

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

```
    trovato=true;
```

```
A++; dim--;
```

```
}
```

iterazioni inutili che si eviterebbero
con

```
while(dim && !trovato) {.....}
```

funzioni ricorsive: per la
correttezza usiamo **l'induzione**

prova induttiva (testo 10.2.1):

-caso base:

PRE<caso base> POST

-passo induttivo:

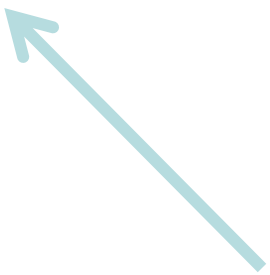
-ipotesi induttiva: le invocazioni
ricorsive sono corrette rispetto
a PRE e POST

-vale PRE <caso non base> POST

ESERCIZIO:

leggere da cin una sequenza di caratteri (che contiene sempre la sentinella ';') e inserirli in posizioni contigue di $A[\text{lim}]$ fino a che c'è posto oppure fino a quando si legge ';' che va anch'esso inserito. Restituire il numero di caratteri inseriti in A

```
int ins(char *A, int p, int lim) //PRE(cin contiene b1...bk ; k>=0)
{
    if(p==lim)
        return 0;
    char x;
    cin >> x;
    if (x==';')
        {A[p]=x; return 1;}
    else
    {
        A[p]=x;
        return 1+ins(A,p+1,lim);
    }
} //POST=(k >= lim-p => A[p..lim-1]=b1..b(lim-p) restituisce lim-p,
altrimenti => A[p..p+k]=b1...bk ;, restituisce k+1)
```



p indice di scorrimento
dell'array

variazioni: inserire solo se nuovo

```
if (! presente(A,p,x))
```

```
    {A[p]=x; return 1+ins(A,p+1,dim);}
```

```
else
```

```
    {return ins(A,p,dim);}
```


altro esercizio: cercare un valore y
in un array A **ordinato**

se c'è, restituire la sua posizione

la funzione deve restituire 2
risultati: bool col return e indice
per riferimento

PRE=(A[0..top-1] ordinato, $0 \leq \text{pos} \leq \text{top}$,)

bool presente(int* A, int top, int pos, int y,
int & start)

POST=

(restituisce true \rightarrow $\text{pos} \leq \text{start} < \text{top}$,
 $A[\text{start}] = y$, $A[\text{pos}..\text{start}-1] < y$) &&
(restituisce false $\rightarrow A[\text{pos}..\text{top}-1] \neq y$)

```
bool presente(int* A, int top, int pos, int y,  
int & start)  
{if(pos==top) return false;  
  
if(A[pos]==y)  
    {start=pos; return true;}  
  
if(A[pos]<y)  
    return presente(A,top,pos+1,y, start);  
  
return false;  
}
```

correttezza per induzione:

-BASE dell'INDUZIONE

dimostrare che se vale la PRE allora
vale la POST nei casi base;

FARE !

-PASSO INDUTTIVO: fare l'ipotesi
che le invocazioni ricorsive
siano corrette rispetto alla PRE e
POST

e poi dimostrare che il corpo è tale
che prima delle invocazioni ricorsive
vale la PRE (dall'ipotesi → vale la
POST al ritorno)

e da questo dimostrare che vale la
POST alla fine del corpo

calcolo di quante occorrenze di y ci sono

PRE= ($A[start..top-1]$ ordinato $\&\&$

$0 \leq start \leq top$ $\&\&$

se $start < top \Rightarrow A[start] \geq y$)

POST= (restituisce k t.c.

$A[start..start+k-1] = y$

$\&\& A[start+k..top-1] > y$)

```
int quanti(int* A, int start, int top, int y)
{
    if(start==top)
        return 0;

    if(A[start]==y)
        return 1+quanti(A,start+1,top,y);

    return 0;
}
```

verifica di
correttezza

eliminare una porzione $A[s..s+q-1]$ di un array

Notazione: se $\underline{A}=[a_0..a_{lim-1}]$, allora
 $\underline{A}-(start,q)=[a_0..a_{start-1}, start+q..a_{lim-1}]$

PRE=($A[lim]$ def. e uguale a $\underline{A}[lim]$, $q>0$,
 $0 \leq s \leq lim-q$)

POST=($A[p..lim-q-1]=\underline{A}-(start,q)[p..lim-q-1]$)


```
void elim(int* A, int lim, int p, int q)
{
    if(p < lim - q)
    {
        A[p] = A[p + q];
        elim(A, lim, p + 1, q);
    }
}

// invocazione iniziale
elim(A, lim, start, quanti);
```

```
void F(int*A, int &lim, int y)
{
    int start=0;
    if(presente(A,lim,0, y,start))
    {
        int quanti=quanti(A, start,lim,
y);
        elim(A,lim,start, quanti);
        lim=lim-q;
    }
} //POST=(elimina gli y da A )
```

da while a ricorsione

input:

x_1, x_2, \dots, x_{n-1}

leggo primo valore e se $\neq -1$ stampo
e poi ricomincio con

x_2, \dots, x_{n-1}

sotto-problema

```
cin >> x;  
while( x != -1)  
{  
  
cout<< x;  
cin >> x;  
}
```



```
cin >> x;  
if( x != -1)  
{  
  
cout<< x;  
cin >> x;  
while( x != -1){  
    cout<< x;  
    cin >> x;  
}  
}}
```

```
void ciclo(int x)
{
```

```
if(x != -1)
```

```
{
```

```
cout<< x;
```

```
cin >> x;
```

```
ciclo(x);
```

```
}}
```

```
if( x != -1)
```

```
{
```

```
cout<< x;
```

```
cin >> x;
```

```
while( x != -1){
```

```
cout<< x;
```

```
cin >> x;
```

```
}}
```

invocazione iniziale:

```
cin >> x;
```

```
ciclo(x);
```

e se facessimo ?

```
void ciclo(int x)
{
    if(x != -1 )
    {int y; cin >> y; ciclo(y); cout<<x;}
}
```


TEORIA

(i) assumendo che l'R-valore di A , dichiarata `int A[4][3][6]`, sia 1000, rispondere alle seguenti domande

-che tipo ha A ?

-che valore e che tipo ha l'espressione $*(A+3)-10$?

-che valore e che tipo ha l'espressione $*(*A+2)-10$.

(ii) Dite se la funzione F e la sua successiva invocazione sono corrette oppure no e spiegate perché (risposte senza spiegazione non sono accettate):

```
int* F(int **x){int* y; y=*x; return *x;}
```

```
main(){int a=9,*p=&a; *F(&p)=a*3;}
```

Teoria

- (i) Si consideri il frammento di programma:

```
int y=1, *p=&y;
```

```
char c=static_cast<char>(p);
```

spiegate brevemente cosa fa
e se è un'operazione corretta
o no.

(ii) Si consideri il seguente programma e si spieghi se è corretto o no motivando la risposta:

```
int*& f(int** z) {int a=1; *z=&a;  
return *z;}
```

```
main(){int a=2, *p=&a, **q=&p;  
*f(q)=*p+a;}
```

Teoria

(1) Dato l'array `char X[10][5][10]`, rispondere ai seguenti due punti:

(i) che tipo ha `*(*X-4)+2` e che differenza c'è tra il suo valore e quello di `X`;

(ii) che tipo ha `X[-1]` e che differenza c'è tra il suo valore e quello di `X`;