



Funzioni ricorsive

Fondamenti di Informatica, AA 2022/23

Luca Cassano

luca.cassano@polimi.it



Richiami sulle funzioni



Un esempio

```
int fattoriale(int n) {  
    int i, f;  
    f = 1;  
    for (i = 2; i <= n; ++i)  
        f = f * i;  
    return f;  
}
```



Un esempio

```
int fattoriale(int n) {
```

```
    int i, f;
```

```
    f = 1;
```

```
    for (i = 2; i <= n; ++i)
```

```
        f = f * i;
```

```
    return f;
```

```
}
```

Parametro formale in ingresso

Uscita della funzione

Definizione di $n!$

Il fattoriale di un intero $n > 0$ è definito come segue:

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



Un esempio di chiamata

```
int fatt;  
fatt = fattoriale(2);
```

```
int fattoriale(int n) {  
    int i, f;  
    f = 1;  
    for (i = 2; i <= n; ++i)  
        f = f * i;  
    return f;  
}
```



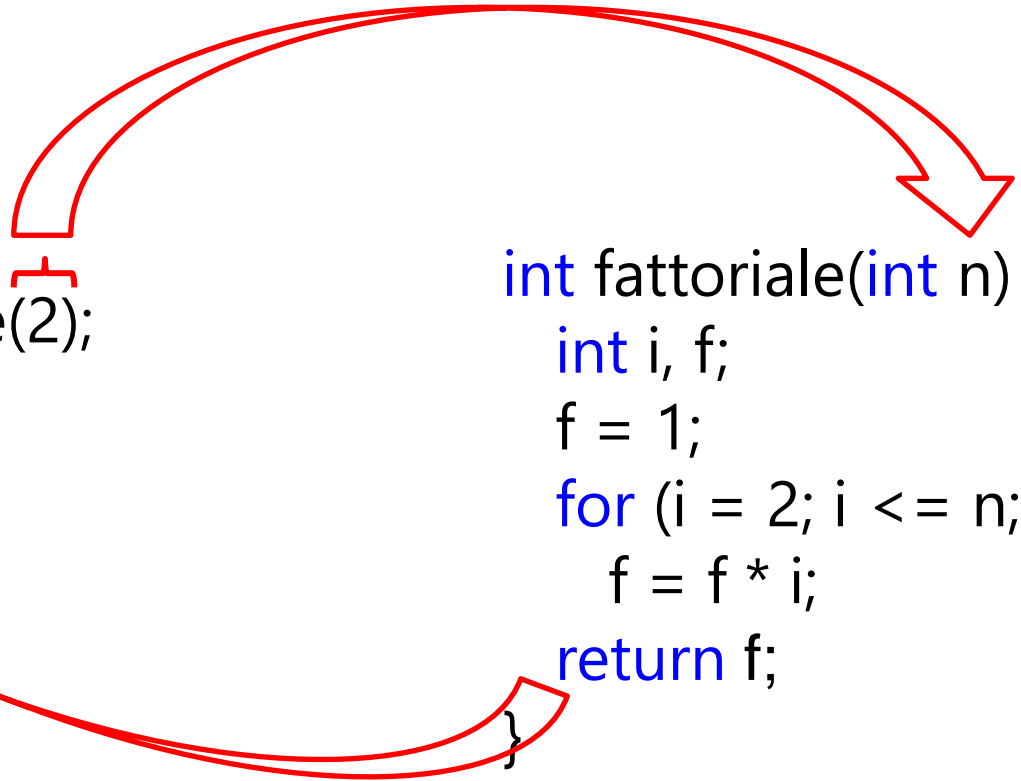
Un esempio di chiamata

Parametro attuale in ingresso

```
int fatt;  
fatt = fattoriale(2);
```

```
int fattoriale(int n) {  
    int i, f;  
    f = 1;  
    for (i = 2; i <= n; ++i)  
        f = f * i;  
    return f;  
}
```

Uscita





Un esempio di chiamata

```
int fatt;  
fatt = fattoriale(2);
```

Record di attivazione chiamante

- Da qui si invoca la funzione
- Contiene la variabile fatt
- Non ha visibilità di f,i,n



Un esempio di chiamata

```
int fatt;  
fatt = fattoriale(2);
```

Record di attivazione chiamante

- Da qui si invoca la funzione
- Contiene la variabile fatt
- Non ha visibilità di f,i,n

```
int fattoriale(int n) {  
    int i, f;  
    f = 1;  
    for (i = 2; i <= n; ++i)  
        f = f * i;  
    return f;  
}
```

Record di attivazione chiamato

- Creato al momento dell'invocazione di fattoriale



Un esempio di chiamata

```
int fatt;  
fatt = fattoriale(2);
```

Record di attivazione chiamante

- Da qui si invoca la funzione
- Contiene la variabile fatt
- Non ha visibilità di f,i,n

```
int fattoriale(int n) {  
    int i, f;  
    f = 1;  
    for (i = 2; i <= n; ++i)  
        f = f * i;  
    return f;  
}
```

Record di attivazione chiamato

- Creato al momento dell'invocazione di fattoriale
- Contiene le variabili n, f, i



Un esempio di chiamata

```
int fatt;  
fatt = fattoriale(2);
```

Record di attivazione chiamante

- Da qui si invoca la funzione
- Contiene la variabile fatt
- Non ha visibilità di f,i,n

```
int fattoriale(int n) {  
    int i, f;  
    f = 1;  
    for (i = 2; i <= n; ++i)  
        f = f * i;  
    return f;  
}
```

Record di attivazione chiamato

- Creato al momento dell'invocazione di fattoriale
- Contiene le variabili n, f, i
- Non ha visibilità su fatt



Un esempio di chiamata

```
int fatt;  
fatt = fattoriale(2);
```

Record di attivazione chiamante

- Da qui si invoca la funzione
- Contiene la variabile fatt
- Non ha visibilità di f,i,n

```
int fattoriale(int n) {  
    int i, f;  
    f = 1;  
    for (i = 2; i <= n; ++i)  
        f = f * i;  
    return f;  
}
```

Record di attivazione chiamato

- Creato al momento dell'invocazione di fattoriale
- Contiene le variabili n, f, i
- Non ha visibilità su fatt
- Non ha legami con il RA chiamante se non per i parametri attuali che vengono copiati (sia in ingresso che in uscita)



Un esempio di chiamata

```
int fatt;  
fatt = fattoriale(2);
```

Record di attivazione chiamante

- Da qui si invoca la funzione
- Contiene la variabile fatt
- Non ha visibilità di f,i,n

```
int fattoriale(int n) {  
    int i, f;  
    f = 1;  
    for (i = 2; i <= n; ++i)  
        f = f * i;  
    return f;  
}
```

Record di attivazione chiamato

- Creato al momento dell'invocazione di fattoriale
- Contiene le variabili n, f, i
- Non ha visibilità su fatt
- Non ha legami con il RA chiamante se non per i parametri attuali che vengono copiati (sia in ingresso che in uscita)
- Distrutto terminata l'esecuzione



Esempi

f2 = fattoriale(2)

2

f3 = fattoriale(3)

6

f4 = fattoriale(4)

24

f5 = fattoriale(5)

120

f6 = fattoriale(6)

720



Un esempio di chiamata

```
int x;  
x = f(2)
```

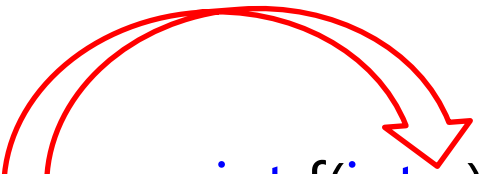
```
int f(int y) {  
    int z;  
    z = g(y);  
    return 2 * z;  
}
```

```
int g(int k)  
    return k / 3;  
}
```



Un esempio di chiamata

```
int x;  
x = f(2)
```



```
int f(int y) {  
    int z;  
    z = g(y);  
    return 2 * z;  
}
```

```
int g(int k)  
    return k / 3;  
}
```

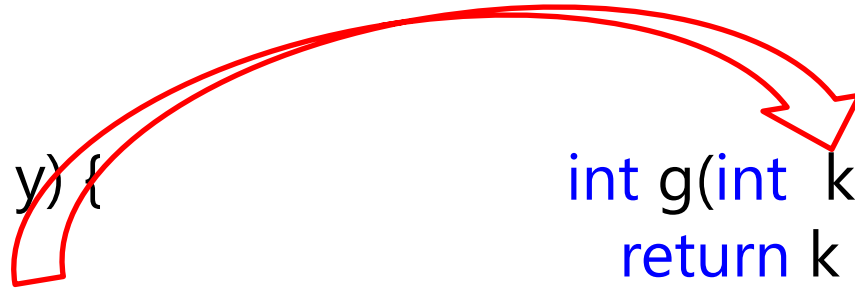


Un esempio di chiamata

```
int x;  
x = f(2)
```

```
int f(int y) {  
    int z;  
    z = g(y);  
    return 2 * z;  
}
```

```
int g(int k)  
    return k / 3;  
}
```



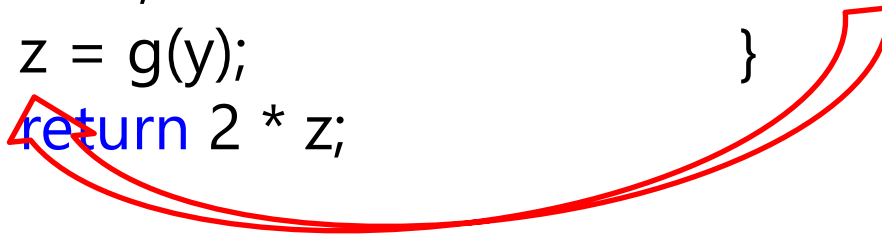


Un esempio di chiamata

```
int x;  
x = f(2)
```

```
int f(int y) {  
    int z;  
    z = g(y);  
    return 2 * z;  
}
```

```
int g(int k)  
    return k / 3;  
}
```





Un esempio di chiamata

```
int x;  
x = f(2)
```

```
int f(int y) {  
    int z;  
    z = g(y);  
    return 2 * z;  
}
```

```
int g(int k)  
    return k / 3;  
}
```



Un esempio

- Si dimostra immediatamente dalla definizione di fattoriale

$$n! = n * \underbrace{(n - 1) * (n - 2) * \cdots * 2 * 1}_{(n - 1)!}$$

- Vale la seguente relazione

$$n! = n * (n - 1)!$$



Un esempio

- Si dimostra immediatamente dalla definizione di fattoriale

$$n! = n * \underbrace{(n - 1) * (n - 2) * \cdots * 2 * 1}_{(n - 1)!}$$

- Vale la seguente relazione

$$n! = n * (n - 1)!$$

- È possibile usare questa proprietà per definire un'implementazione di fattoriale totalmente diversa?



Ricorsione

Funzioni che chiamano se stesse



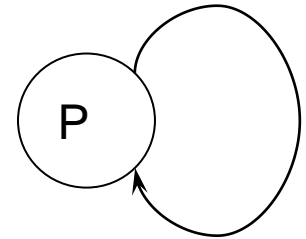
Ricorsione

- Che cos'è la ricorsione?



Ricorsione

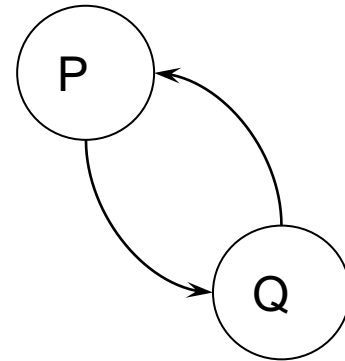
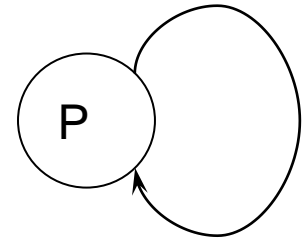
- Che cos'è la ricorsione?
 - Un sottoprogramma P richiama se stesso (ricorsione diretta)





Ricorsione

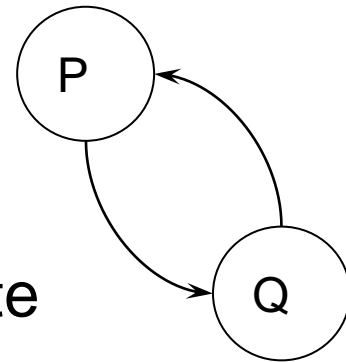
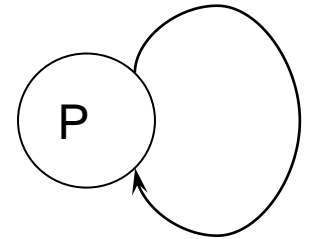
- Che cos'è la ricorsione?
 - Un sottoprogramma P richiama se stesso (ricorsione diretta)
 - Un sottoprogramma P richiama un sottoprogramma Q che comporta un'altra chiamata a P (ricorsione indiretta)





Ricorsione

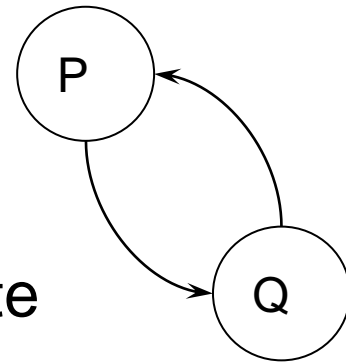
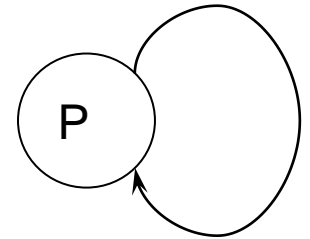
- Che cos'è la ricorsione?
 - Un sottoprogramma P richiama se stesso (ricorsione diretta)
 - Un sottoprogramma P richiama un sottoprogramma Q che comporta un'altra chiamata a P (ricorsione indiretta)
- È una tecnica di programmazione molto potente





Ricorsione

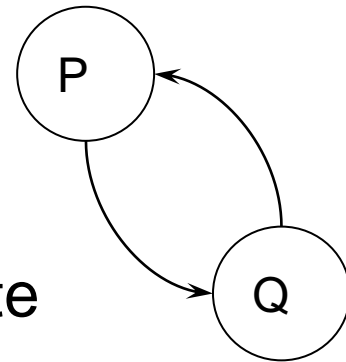
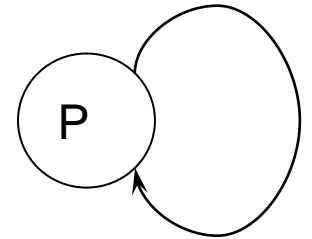
- Che cos'è la ricorsione?
 - Un sottoprogramma P richiama se stesso (ricorsione diretta)
 - Un sottoprogramma P richiama un sottoprogramma Q che comporta un'altra chiamata a P (ricorsione indiretta)
- È una tecnica di programmazione molto potente
- Permette di risolvere in maniera elegante problemi complessi





Ricorsione

- Che cos'è la ricorsione?
 - Un sottoprogramma P richiama se stesso (ricorsione diretta)
 - Un sottoprogramma P richiama un sottoprogramma Q che comporta un'altra chiamata a P (ricorsione indiretta)
- È una tecnica di programmazione molto potente
- Permette di risolvere in maniera elegante problemi complessi
- Le funzioni che richiamano se stesse (direttamente o indirettamente) sono dette **funzioni ricorsive**





Esempio: il fattoriale Ricorsivo

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



Esempio: il fattoriale Ricorsivo

```
int factRic(int n) {
```

```
    if (n == 0)
```

```
        return 1;
```

```
    else
```

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

```
}
```

Questa ci ricorda
 $0! = 1$



Esempio: il fattoriale Ricorsivo

```
int factRic(int n) {
```

```
    if (n == 0)
```

```
        return 1;
```

```
    else
```

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

```
}
```

Questa ci ricorda
 $n! = n * (n - 1)!$



Esempio: il fattoriale Ricorsivo

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



Una funzione che chiama se stessa



Una Funzione che Chiama se Stessa?

- In ogni istante possono essere in corso **diverse attivazioni** (*istanze*) dello **stesso** sottoprogramma



Una Funzione che Chiama se Stessa?

- In ogni istante possono essere in corso **diverse attivazioni** (*istanze*) dello **stesso** sottoprogramma
 - Ovviamente sono **tutte sospese tranne** una, **l'ultima invocata**, all'interno della quale si sta svolgendo il **flusso di esecuzione**.



Una Funzione che Chiama se Stessa?

- In ogni istante possono essere in corso **diverse attivazioni** (*istanze*) dello **stesso** sottoprogramma
 - Ovviamente sono **tutte sospese tranne** una, **l'ultima invocata**, all'interno della quale si sta svolgendo il **flusso di esecuzione**.
- Ogni attivazione esegue **lo stesso codice** ma opera su **record di attivazione distinti**



Una Funzione che Chiama se Stessa?

- In ogni istante possono essere in corso **diverse attivazioni** (*istanze*) dello **stesso** sottoprogramma
 - Ovviamente sono **tutte sospese tranne una, l'ultima invocata**, all'interno della quale si sta svolgendo il **flusso di esecuzione**.
- Ogni attivazione esegue **lo stesso codice** ma opera su **record di attivazione distinti**
 - Si hanno quindi **copie distinte** dei **parametri attuali** e delle **variabili locali** nelle varie invocazioni



Una funzione che chiama se stessa?

- ... se ogni volta la funzione richiama se stessa... *perché la catena di invocazioni non continua **all'infinito**?*



Una funzione che chiama se stessa?

- ... se ogni volta la funzione richiama se stessa... *perché la catena di invocazioni non continua **all'infinito**?*
- La funzione ricorsiva deve prevedere una situazione in cui non richiama se stessa, i.e., il **caso base**



Programmazione Ricorsiva

Per risolvere un problema attraverso la programmazione ricorsiva sono **necessari alcuni elementi**



Programmazione Ricorsiva

Per risolvere un problema attraverso la programmazione ricorsiva sono **necessari alcuni elementi**

- **Caso base:** caso elementare del problema che può essere risolto immediatamente. **Il caso base non deve eseguire alcuna chiamata ricorsiva.**



Programmazione Ricorsiva

Per risolvere un problema attraverso la programmazione ricorsiva sono **necessari alcuni elementi**

- **Caso base:** caso elementare del problema che può essere risolto immediatamente. **Il caso base non deve eseguire alcuna chiamata ricorsiva.**
- **Passo ricorsivo:** chiamata ricorsiva per risolvere uno o più problemi più semplici



Programmazione Ricorsiva

Per risolvere un problema attraverso la programmazione ricorsiva sono **necessari alcuni elementi**

- **Caso base:** caso elementare del problema che può essere risolto immediatamente. **Il caso base non deve eseguire alcuna chiamata ricorsiva.**
- **Passo ricorsivo:** chiamata ricorsiva per risolvere uno o più problemi più semplici
- **Costruzione della soluzione:** costruzione della soluzione sulla base del risultato delle chiamate ricorsive



Programmazione Ricorsiva

Per risolvere un problema attraverso la programmazione ricorsiva sono **necessari alcuni elementi**

- **Caso base:** caso elementare del problema che può essere risolto immediatamente. **Il caso base non deve eseguire alcuna chiamata ricorsiva.**
- **Passo ricorsivo:** chiamata ricorsiva per risolvere uno o più problemi più semplici
- **Costruzione della soluzione:** costruzione della soluzione sulla base del risultato delle chiamate ricorsive

Affinché l'esecuzione della funzione termini è fondamentale che la soluzione ricorsiva converga verso il caso base



Esempio: il fattoriale

- Definizione:
$$f(n) = n! = n * (n-1) * (n-2) * \dots * 3 * 2 * 1$$
- Definire caso base, e passo ricorsivo



Esempio: il fattoriale

- Definizione:
$$f(n) = n! = n * (n-1) * (n-2) * \dots * 3 * 2 * 1$$
- Passo ricorsivo:
$$f(n) = n * f(n-1)$$
- Caso base:
$$f(0) = 1$$



Esempio: il fattoriale

- Definizione:
 $f(n) = n! = n * (n-1) * (n-2) * \dots * 3 * 2 * 1$
- Passo ricorsivo:
 $f(n) = n * f(n-1)$
- Caso base:
 $f(0) = 1$

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



Funzionamento ricorsione

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

X = factRic(3)

factRic

n:3

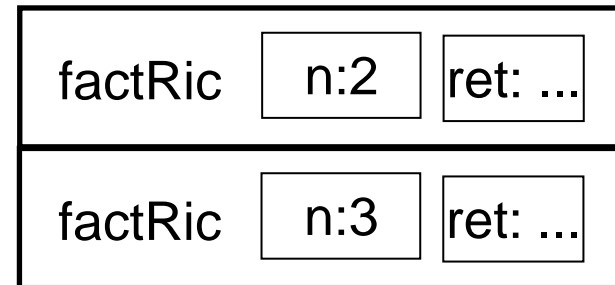
ret: ...



Funzionamento ricorsione

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

X = factRic(3)





Funzionamento ricorsione

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

X = factRic(3)

factRic	n:1	ret: ...
factRic	n:2	ret: ...
factRic	n:3	ret: ...



Funzionamento ricorsione

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

X = factRic(3)

factRic	n:0	ret: ...
factRic	n:1	ret: ...
factRic	n:2	ret: ...
factRic	n:3	ret: ...



Funzionamento ricorsione

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

X = factRic(3)

factRic	n:0	ret: 1
factRic	n:1	ret: ...
factRic	n:2	ret: ...
factRic	n:3	ret: ...



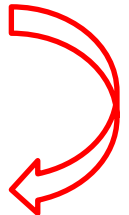


Funzionamento ricorsione

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

X = factRic(3)

factRic	n:0	ret: 1
factRic	n:1	ret: ...
factRic	n:2	ret: ...
factRic	n:3	ret: ...





Funzionamento ricorsione

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

X = factRic(3)

factRic	n:1	ret: 1
factRic	n:2	ret: ...
factRic	n:3	ret: ...

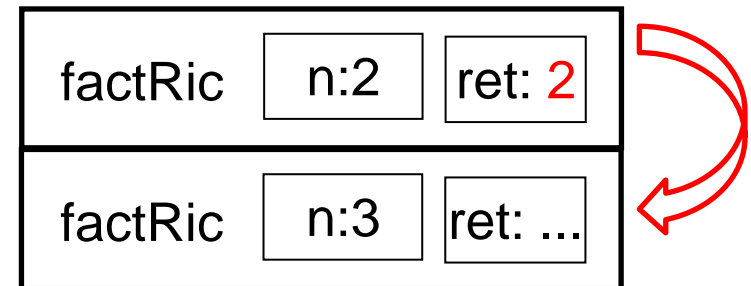




Funzionamento ricorsione

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

X = factRic(3)

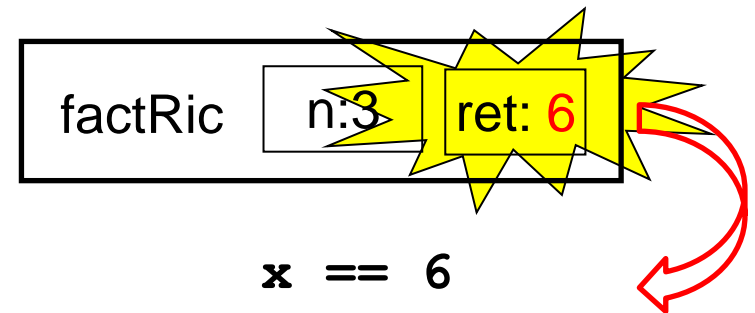




Funzionamento ricorsione

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

x = factRic(3)





Ricorsione in Coda

Ricorsione in cui la funzione:

- prevede **una sola chiamata ricorsiva**
- esegue la chiamata ricorsiva come **ultima istruzione**

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



Problemi nell'Uso della Ricorsione

Terminazione della catena ricorsiva

- È presente il caso base?
- Viene raggiunto sempre dalla catena di chiamate ricorsive?



Errori nell'Uso della Ricorsione

Catena infinita di chiamate incondizionate

- **Deve esistere** una **condizione** tale per cui **non** viene eseguita la **chiamata ricorsiva (caso base)**

```
int factRic(int n) {  
    return n * factRic(n - 1);  
}
```



Errori nell'Uso della Ricorsione

Catena infinita di chiamate incondizionate

- **Deve esistere** una **condizione** tale per cui **non** viene eseguita la **chiamata ricorsiva (caso base)**

```
int factRic(int n) {  
    return n * factRic(n - 1);  
}
```



Manca il
caso base

Es la chiamata a factRic(7) chiama factRic(7),
che chiama factRic(6),
che chiama factRic(5),
che chiama factRic(4),
che chiama factRic(3),....

che chiama factRic(-1000),....



Errori nell'Uso della Ricorsione

Catena infinita di chiamate incondizionate

- **è necessario che questa condizione venga raggiunta:** anche programmi formalmente corretti potrebbero non funzionare per alcuni valori degli ingressi

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



Errori nell'Uso della Ricorsione

Catena infinita di chiamate incondizionate

- **è necessario che questa condizione venga raggiunta:** anche programmi formalmente corretti potrebbero non funzionare per alcuni valori degli ingressi

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

- Ad es, `factRic(-1)` dà luogo ad una sequenza di chiamate infinite



Errori nell'Uso della Ricorsione

Catena infinita di chiamate identiche:

- La chiamata ricorsiva **non** può avere i **parametri formali uguali a quelli attuali**

```
int factRic(int n) {  
    if (n == 0)  
        return 1;  
    else  
        return n * factRic(n);  
}
```

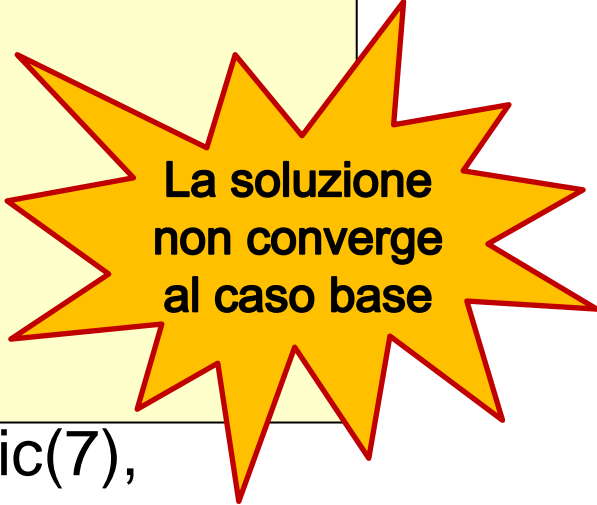


Errori nell'Uso della Ricorsione

Catena infinita di chiamate identiche:

- La chiamata ricorsiva **non** può avere i **parametri formali uguali a quelli attuali**

```
int factRic(int n) {  
    if (n == 0)  
        return 1;  
    else  
        return n * factRic(n);  
}
```



La soluzione
non converge
al caso base

Es la chiamata a factRic(7) chiama factRic(7),
che chiama factRic(7),
che chiama factRic(7),
che chiama factRic(7),
che chiama factRic(7),....



Condizioni Necessarie

.. Per evitare ricorsione infinita:

- Occorre che le **chiamate ricorsive** siano **soggette** a una **condizione** che prima o poi assicura che la catena termini
- Occorre anche che **l'argomento sia *progressivamente modificato*** dal passo ricorsivo, in modo da tendere prima o poi al caso base
 - Nella pratica l'argomento si avvicina al valore nel caso base



Esempio

Scrivere una funzione ricorsiva che calcola la somma di tutti gli interi compresi tra due argomenti passati come parametri



```
int sommaNumeriCompresi(int a , int b) {
```



Esempio

Scrivere una funzione ricorsiva che calcola la somma di tutti gli interi compresi tra due argomenti passati come parametri

```
int sommaNumeriCompresi(int a , int b) {  
    if (      ) {  
  
    }  
    else {  
  
    }  
}
```



Esempio

Scrivere una funzione ricorsiva che calcola la somma di tutti gli interi compresi tra due argomenti passati come parametri

```
int sommaNumeriCompresi(int a , int b) {  
    if (a == b) {  
  
    }  
    else {  
  
    }  
}
```



Esempio

Scrivere una funzione ricorsiva che calcola la somma di tutti gli interi compresi tra due argomenti passati come parametri

```
int sommaNumeriCompresi(int a , int b) {  
    if (a == b) {  
        return a;  
    }  
    else {  
  
    }  
}
```



Esempio

Scrivere una funzione ricorsiva che calcola la somma di tutti gli interi compresi tra due argomenti passati come parametri

```
int sommaNumeriCompresi(int a , int b) {  
    if (a == b) {  
        return a;  
    }  
    else {  
        return a + sommaNumeriCompresi(a+1 , b);  
    }  
}
```



sommaNumeriCompresi(3 ,6)

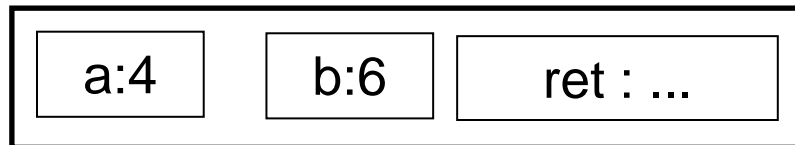
a:3

b:6

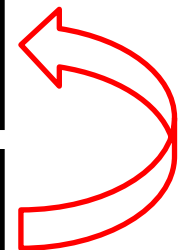
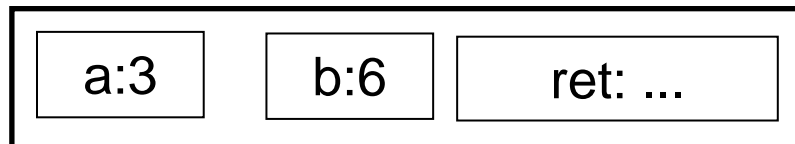
ret: ...



sommaNumeriCompresi(4 ,6)



sommaNumeriCompresi(3 ,6)





sommaNumeriCompresi(5 ,6)

a:5

b:6

ret : ...

sommaNumeriCompresi(4 ,6)

a:4

b:6

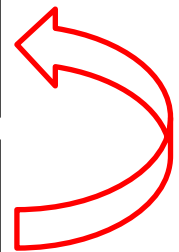
ret : ...

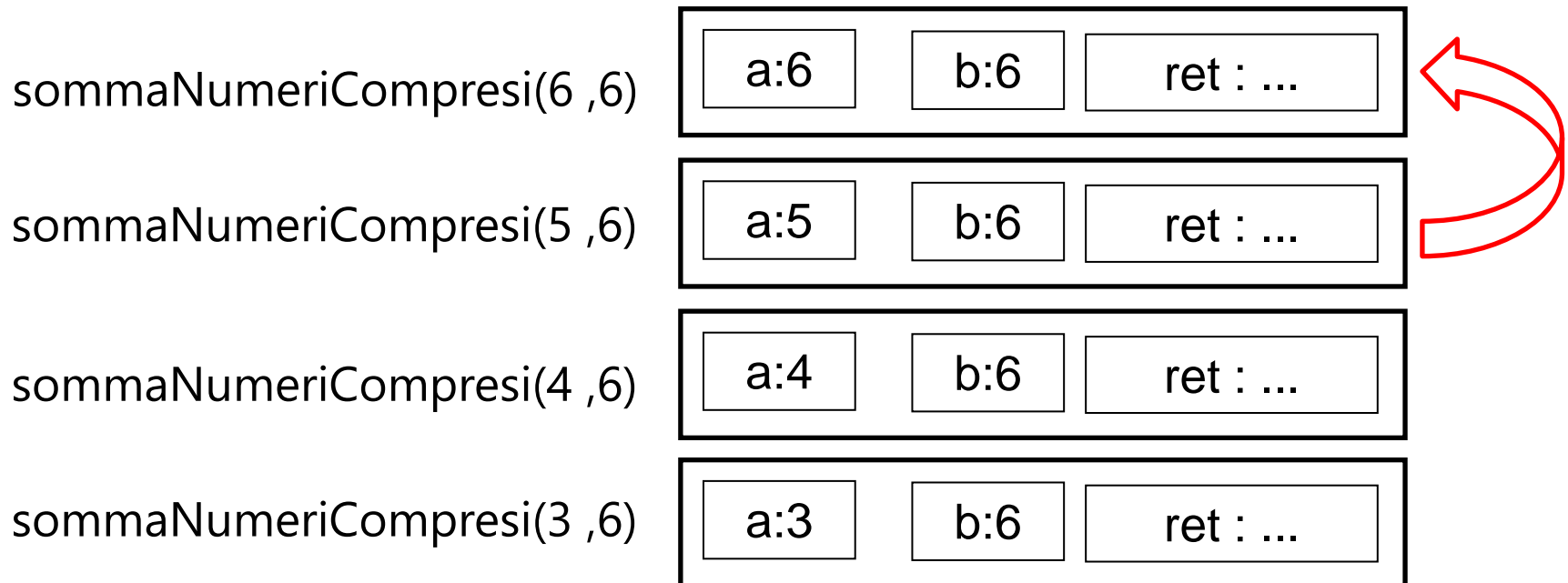
sommaNumeriCompresi(3 ,6)

a:3

b:6

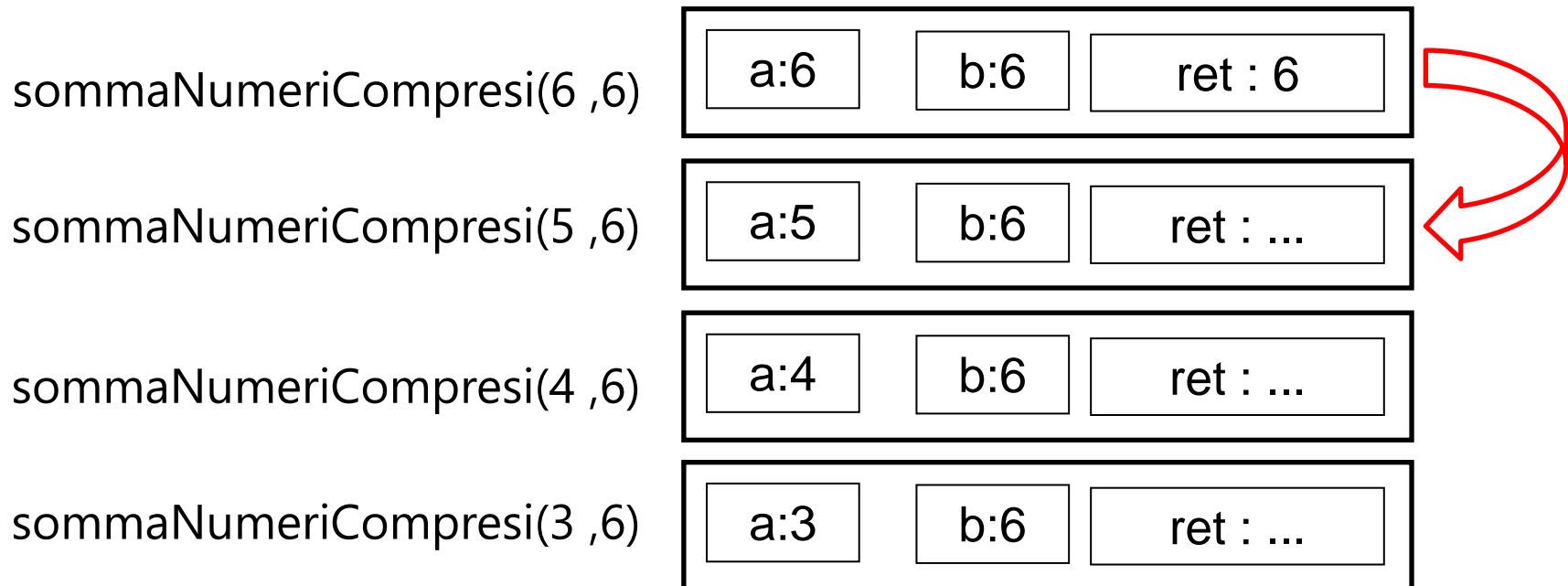
ret : ...







sommaNumeriCompresi(6 ,6)	a:6	b:6	ret : 6
sommaNumeriCompresi(5 ,6)	a:5	b:6	ret : ...
sommaNumeriCompresi(4 ,6)	a:4	b:6	ret : ...
sommaNumeriCompresi(3 ,6)	a:3	b:6	ret : ...





sommaNumeriCompresi(5 ,6)

a:5

b:6

ret : 11

sommaNumeriCompresi(4 ,6)

a:4

b:6

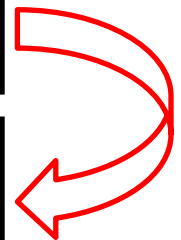
ret : ...

sommaNumeriCompresi(3 ,6)

a:3

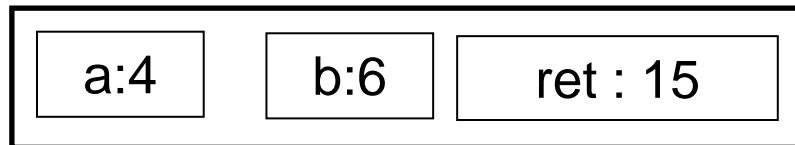
b:6

ret : ...

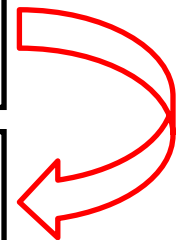
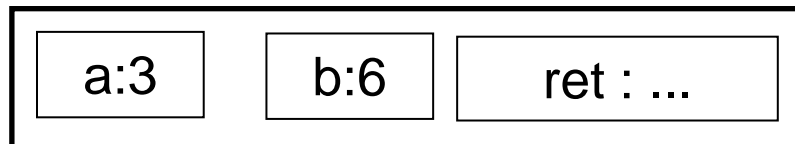




sommaNumeriCompresi(4 ,6)



sommaNumeriCompresi(3 ,6)





sommaNumeriCompresi(3 ,6)

a:3

b:6

ret : 18



Qual è il problema nella soluzione proposta?



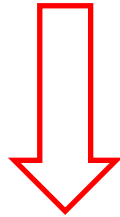
Qual è il problema nella soluzione proposta?

Cosa succede se $a > b$?



Qual è il problema nella soluzione proposta?

Cosa succede se $a > b$?



Chiamate ricorsive infinite



Esempio

Scrivere una funzione ricorsiva che calcola la somma di tutti gli interi compresi tra due argomenti passati come parametri

```
int sommaNumeriCompresi(int a, int b) {  
    if (a > b) {  
        return 0;  
    }  
    if (a == b) {  
        return a;  
    }  
    else {  
        return a + sommaNumeriCompresi(a+1 , b);  
    }  
}
```



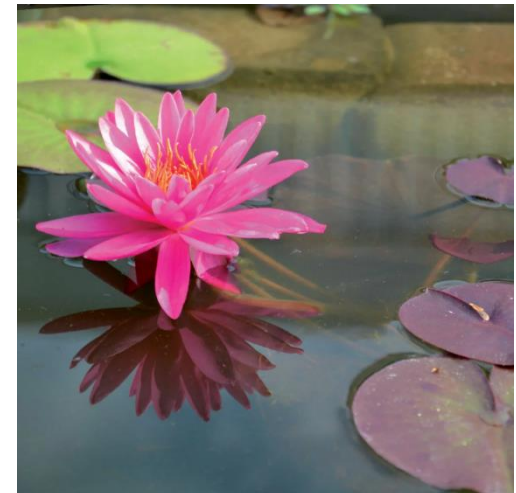
Scrivere una funzione ricorsiva che calcola la somma di tutti gli interi compresi tra due argomenti passati come parametri

```
int sommaNumeriCompresi(int a, int b) {  
    int c;  
    if (a > b) {  
        c = a;  
        a = b;  
        b = c;  
    }  
    if (a == b) {  
        return a;  
    }  
    else {  
        return a + sommaNumeriCompresi(a+1 , b);  
    }  
}
```



Le ninfee

- Uno stagno contiene n ninfee
 - Ogni ninfea in 1 giorno si riproduce di un fattore f (se $f == 2$ ogni ninfea ogni giorno genera un'altra ninfea, se $f == 3$ ogni ninfea ogni giorno genera due ninfee...)
- Scrivere una funzione ricorsiva per calcolare in quanti giorni la popolazione iniziale di n ninfee ha raggiunto una cardinalità di almeno N individui dato un fattore f fissato





Soluzione

```
int stagno(int n, double f, int N) {
```

```
}
```



Soluzione

```
int stagno(int n, double f, int N) {  
    /* caso base: se ho già almeno N ninfee nello stagno */  
    /* non devo aspettare alcun giorno */  
    if (      ) {  
  
    }  
    else {  
  
  
    }  
}
```



Soluzione

```
int stagno(int n, double f, int N) {  
    /* caso base: se ho già almeno N ninfee nello stagno */  
    /* non devo aspettare alcun giorno */  
    if (n >= N) {  
        return 0;  
    }  
    else {  
  
    }  
}
```



Soluzione

```
int stagno(int n, double f, int N) {  
    /* caso base: se ho già almeno N ninfee nello stagno */  
    /* non devo aspettare alcun giorno */  
    if (n >= N) {  
        return 0;  
    }  
    else {  
        /* non abbiamo ancora N ninfee -> chiamata ricorsiva: */  
        /* il numero di giorni necessari per coprire lo stagno è */  
        /* uguale a: un giorno + */  
        /* il numero di giorni che saranno necessari domani */  
        /* (quando le ninfee saranno diventate n*f). */  
  
    }  
}
```




Soluzione

```
int stagno(int n, double f, int N) {  
    /* caso base: se ho già almeno N ninfee nello stagno */  
    /* non devo aspettare alcun giorno */  
    if (n >= N) {  
        return 0;  
    }  
    else {  
        /* non abbiamo ancora N ninfee -> chiamata ricorsiva: */  
        /* il numero di giorni necessari per coprire lo stagno è */  
        /* uguale a: un giorno + */  
        /* il numero di giorni che saranno necessari domani */  
        /* (quando le ninfee saranno diventate n*f). */  
        return 1 + stagno(n * f, f, N);  
    }  
}
```



Scrivere una funzione ricorsiva per calcolare la lunghezza di una stringa



```
int calcolaLunghezza(char *str) {
```



Scrivere una funzione ricorsiva per calcolare la lunghezza di una stringa

```
int calcolaLunghezza(char *str) {  
    if(          ) {  
  
    }  
    else {  
  
    }  
}
```



Scrivere una funzione ricorsiva per calcolare la lunghezza di una stringa

```
int calcolaLunghezza(char *str) {  
    if(*str == '\0') {  
        return 0;  
    }  
    else {  
  
    }  
}
```



Scrivere una funzione ricorsiva per calcolare la lunghezza di una stringa

```
int calcolaLunghezza(char *str) {  
    if(*str == '\0') {  
        return 0;  
    }  
    else {  
        return 1 + calcolaLunghezza(str+1);  
    }  
}
```



calcolaLunghezza('ciao')

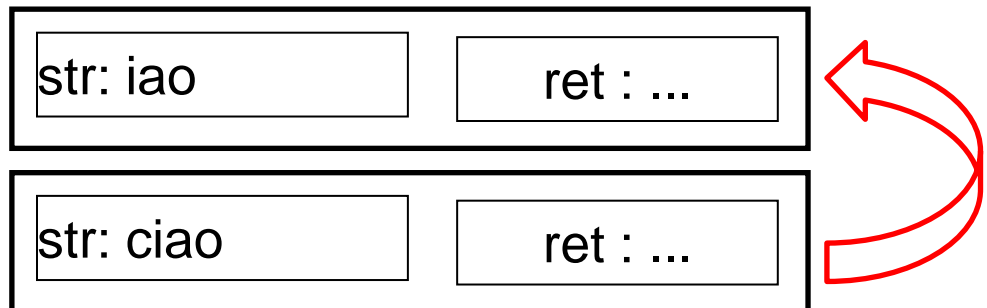
str: ciao

ret: ...



calcolaLunghezza('iao')

calcolaLunghezza('ciao')





calcolaLunghezza('ao')

str: ao

ret : ...

calcolaLunghezza('iao')

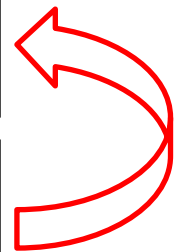
str: iao

ret : ...

calcolaLunghezza('ciao')

str: ciao

ret : ...





calcolaLunghezza('o')

str: o

ret : ...

calcolaLunghezza('ao')

str: ao

ret : ...

calcolaLunghezza('iao')

str: iao

ret : ...

calcolaLunghezza('ciao')

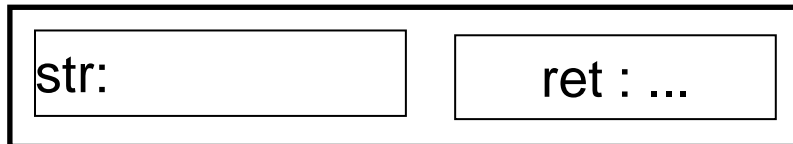
str: ciao

ret : ...

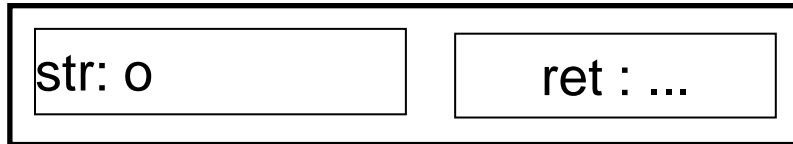




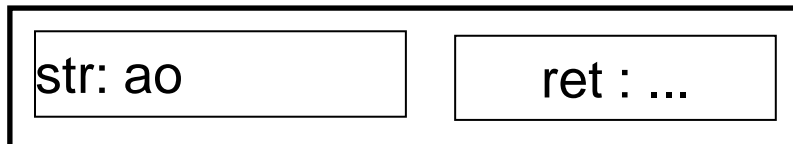
calcolaLunghezza("")



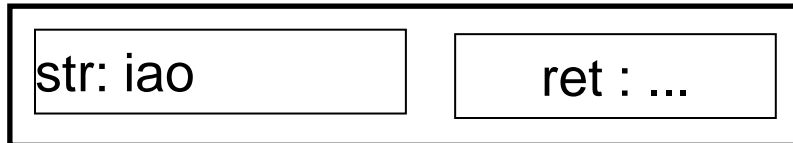
calcolaLunghezza('o')



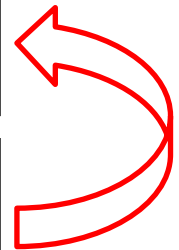
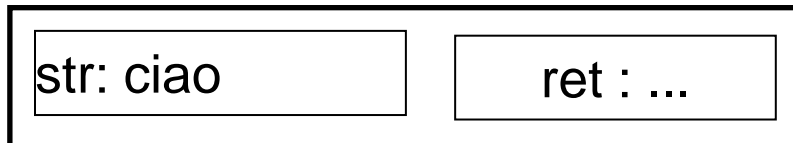
calcolaLunghezza('ao')



calcolaLunghezza('iao')



calcolaLunghezza('ciao')





calcolaLunghezza("")

str:

ret : 0

calcolaLunghezza('o')

str: o

ret : ...

calcolaLunghezza('ao')

str: ao

ret : ...

calcolaLunghezza('iao')

str: iao

ret : ...

calcolaLunghezza('ciao')

str: ciao

ret : ...



calcolaLunghezza("")

str:

ret : 0

calcolaLunghezza('o')

str: o

ret : ...

calcolaLunghezza('ao')

str: ao

ret : ...

calcolaLunghezza('iao')

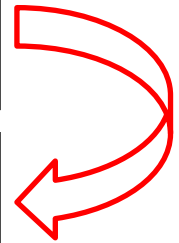
str: iao

ret : ...

calcolaLunghezza('ciao')

str: ciao

ret : ...





calcolaLunghezza('o')

str: o

ret : 1

calcolaLunghezza('ao')

str: ao

ret : ...

calcolaLunghezza('iao')

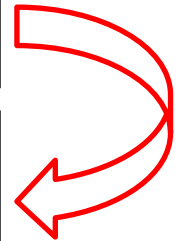
str: iao

ret : ...

calcolaLunghezza('ciao')

str: ciao

ret : ...





calcolaLunghezza('ao')

str: ao

ret : 2

calcolaLunghezza('iao')

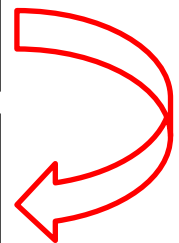
str: iao

ret : ...

calcolaLunghezza('ciao')

str: ciao

ret : ...



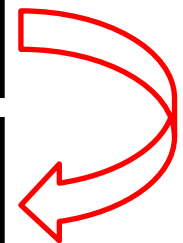


calcolaLunghezza('iao')

calcolaLunghezza('ciao')

str: iao	ret : 3
----------	---------

str: ciao	ret : ...
-----------	-----------





calcolaLunghezza('ciao')

str: ciao

ret : 4



Scrivere una funzione per stampare una stringa al contrario



```
void stampaAlContrario(char *frase) {
```

```
void stampaAlContrario(char *frase) {
```



```
void stampaAlContrario(char *frase) {  
    /* caso base: se sono arrivato alla fine della stringa  
    non faccio nulla */  
  
}
```



Esempio:

Scrivere una funzione per stampare una stringa al contrario

```
void stampaAlContrario(char *frase) {  
    /* caso base: se sono arrivato alla fine della stringa  
    non faccio nulla */  
    /* altrimenti */  
    if (          ) {  
  
    }  
}
```



Esempio:

Scrivere una funzione per stampare una stringa al contrario

```
void stampaAlContrario(char *frase) {  
    /* caso base: se sono arrivato alla fine della stringa  
    non faccio nulla */  
    /* altrimenti */  
    if (*frase != '\0') {  
  
        }  
    }  
}
```



Esempio:

Scrivere una funzione per stampare una stringa al contrario

```
void stampaAlContrario(char *frase) {  
    /* caso base: se sono arrivato alla fine della stringa  
    non faccio nulla */  
    /* altrimenti */  
    if (*frase != '\0') {  
        stampaAlContrario(frase+1);  
        printf("%c", *frase);  
    }  
}
```



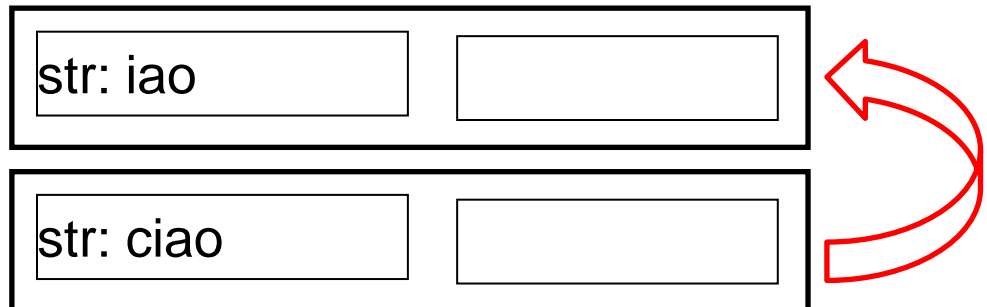
stampaAlContrario('ciao')

str: ciao



stampaAlContrario('iao')

stampaAlContrario('ciao')

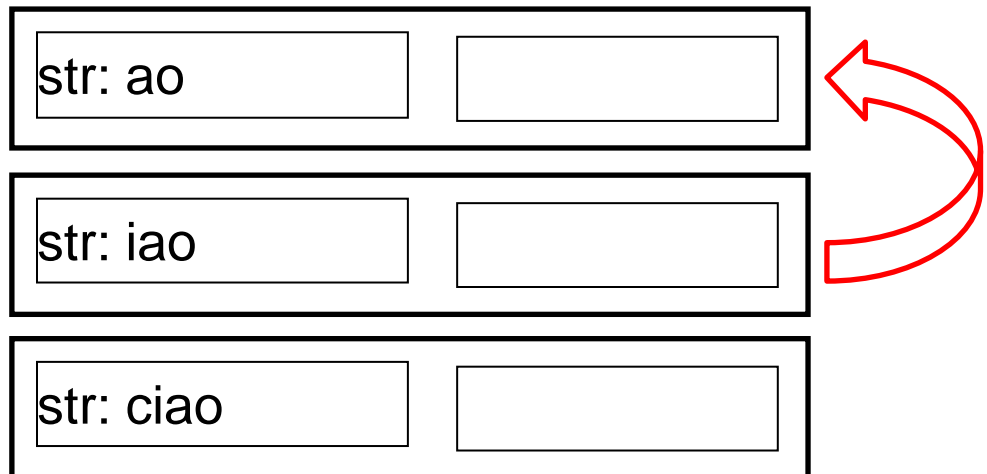




stampaAlContrario('ao')

stampaAlContrario('iao')

stampaAlContrario('ciao')





stampaAlContrario('o')

str: o

stampaAlContrario('ao')

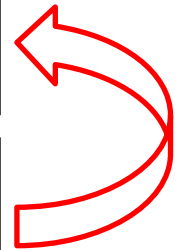
str: ao

stampaAlContrario('iao')

str: iao

stampaAlContrario('ciao')

str: ciao





stampaAlContrario("")

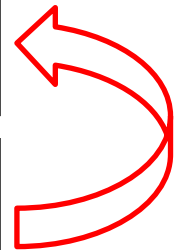
stampaAlContrario('o')

stampaAlContrario('ao')

stampaAlContrario('iao')

stampaAlContrario('ciao')

str:	
str: o	
str: ao	
str: iao	
str: ciao	





stampaAlContrario("")

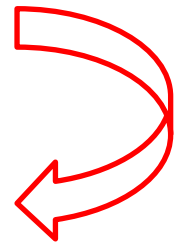
stampaAlContrario('o')

stampaAlContrario('ao')

stampaAlContrario('iao')

stampaAlContrario('ciao')

str:	
str: o	
str: ao	
str: iao	
str: ciao	





stampaAlContrario('o')

str: o

o

stampaAlContrario('ao')

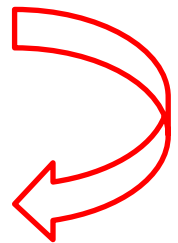
str: ao

stampaAlContrario('iao')

str: iao

stampaAlContrario('ciao')

str: ciao





stampaAlContrario('ao')

str: ao

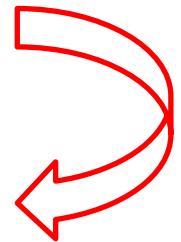
a

stampaAlContrario('iao')

str: iao

stampaAlContrario('ciao')

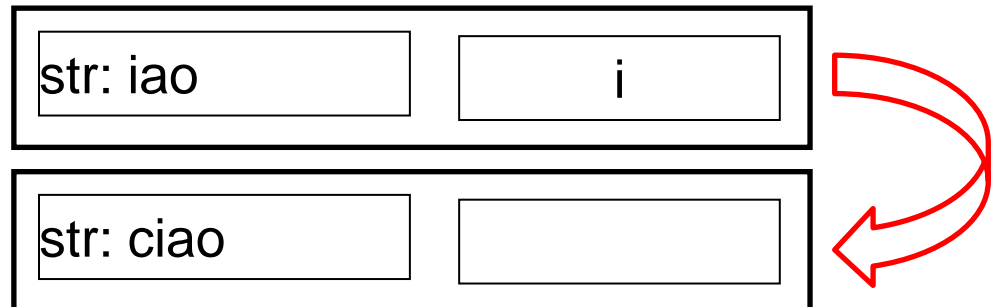
str: ciao





stampaAlContrario('iao')

stampaAlContrario('ciao')





stampaAlContrario('ciao')

str: ciao

c



Esempio:

Modificare la funzione precedente per stampare la stringa nello stesso ordine in cui è stata inserita



Esempio:

Modificare la funzione precedente per stampare la stringa nello stesso ordine in cui è stata inserita

```
void stampaAlContrario(char *frase) {  
    /* caso base: se sono arrivato alla fine della stringa  
    non faccio nulla */  
    /* altrimenti */  
    if (*frase != '\0') {  
        printf("%c", *frase);  
        stampaAlContrario(frase+1);  
    }  
}
```



stampaAlContrario('ciao')

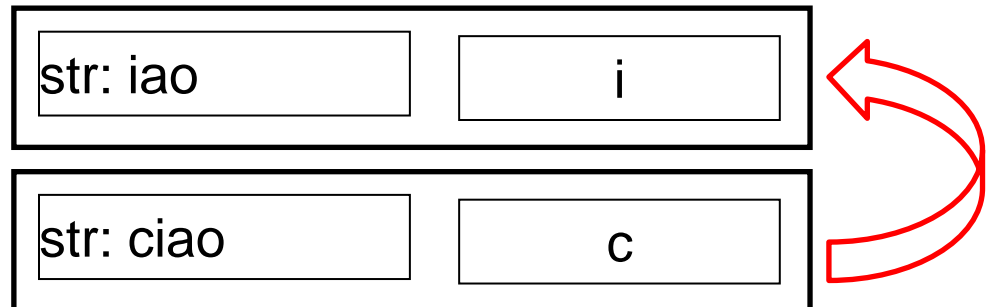
str: ciao

c



stampaAlContrario('cia')

stampaAlContrario('ciao')

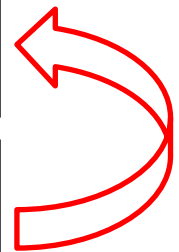
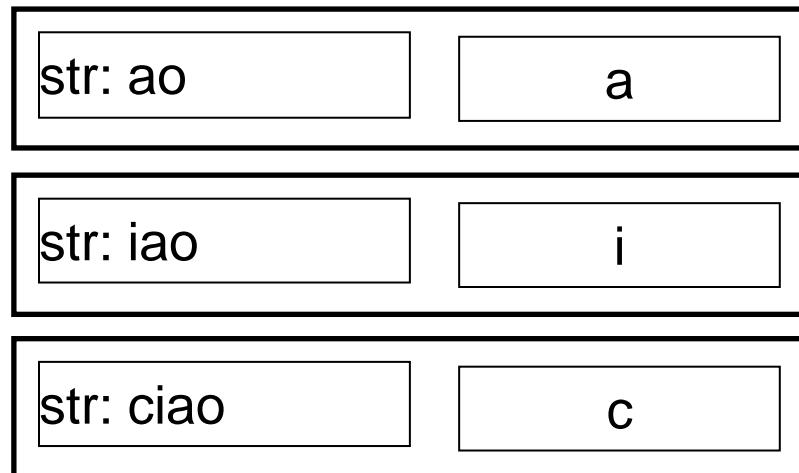




stampaAlContrario('ci')

stampaAlContrario('cia')

stampaAlContrario('ciao')





stampaAlContrario('c')

str: o

o

stampaAlContrario('ci')

str: ao

a

stampaAlContrario('cia')

str: iao

i

stampaAlContrario('ciao')

str: ciao

c





stampaAlContrario('')

str:	
------	--

stampaAlContrario('c')

str: o	o
--------	---

stampaAlContrario('ci')

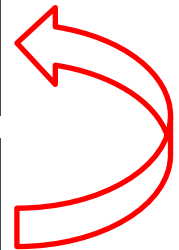
str: ao	a
---------	---

stampaAlContrario('cia')

str: iao	i
----------	---

stampaAlContrario('ciao')

str: ciao	c
-----------	---





stampaAlContrario("")

str:	
------	--

stampaAlContrario('c')

str: o	o
--------	---

stampaAlContrario('ci')

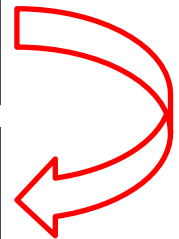
str: ao	a
---------	---

stampaAlContrario('cia')

str: iao	i
----------	---

stampaAlContrario('ciao')

str: ciao	c
-----------	---





stampaAlContrario('c')

str: o

o

stampaAlContrario('ci')

str: ao

a

stampaAlContrario('cia')

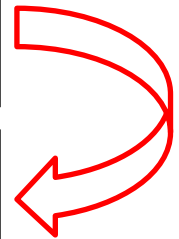
str: iao

i

stampaAlContrario('ciao')

str: ciao

c

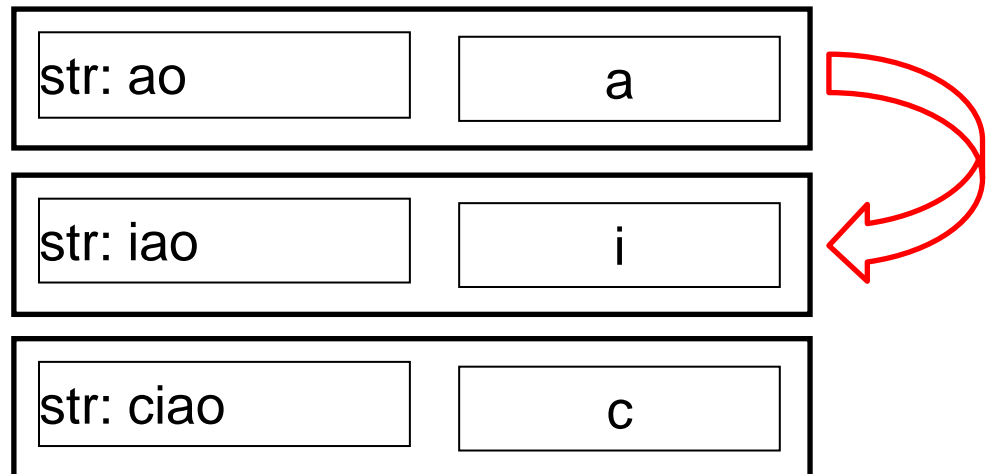




stampaAlContrario('ci')

stampaAlContrario('cia')

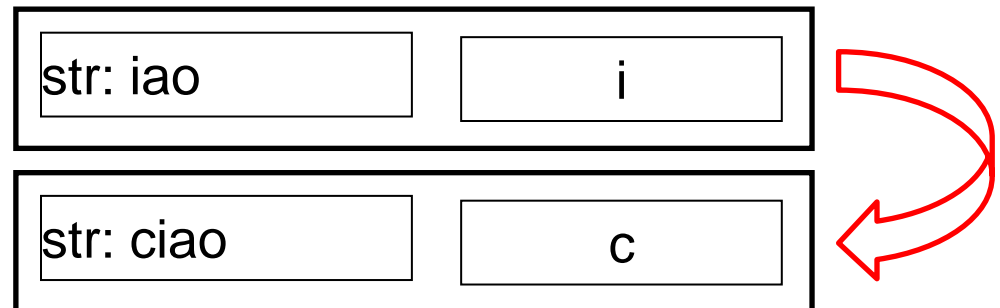
stampaAlContrario('ciao')





stampaAlContrario('cia')

stampaAlContrario('ciao')





stampaAlContrario('ciao')

str: ciao

c



Definizione ricorsiva della successione di Fibonacci

I numeri di Fibonacci

- Modello alla base di molte dinamiche evolutive delle popolazioni

$$F = \{f_1, \dots, f_n\}$$

- $f_1 = 1$
- $f_2 = 1$
- Per $n > 2$, $f_n = f_{n-1} + f_{n-2}$



Definizione ricorsiva della successione di Fibonacci

I numeri di Fibonacci

- Modello alla base di molte dinamiche evolutive delle popolazioni

$$F = \{f_1, \dots, f_n\}$$

- $f_1 = 1$
 - $f_2 = 1$
 - Per $n > 2$, $f_n = f_{n-1} + f_{n-2}$
- } casi base (sono 2!)



Definizione ricorsiva della successione di Fibonacci

I numeri di Fibonacci

- Modello alla base di molte dinamiche evolutive delle popolazioni

$$F = \{f_1, \dots, f_n\}$$

- $f_1 = 1$
 - $f_2 = 1$
 - Per $n > 2$, $f_n = f_{n-1} + f_{n-2}$
- } casi base (sono 2!)
- } 1 passo ricorsivo che contiene due chiamate ricorsive



Definizione ricorsiva della successione di Fibonacci

I numeri di Fibonacci

- Modello alla base di molte dinamiche evolutive delle popolazioni

F(4)

$$F = \{f_1, \dots, f_n\}$$

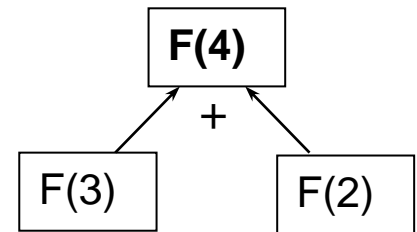
- $f_1 = 1$
 - $f_2 = 1$
 - Per $n > 2$, $f_n = f_{n-1} + f_{n-2}$
- } casi base (sono 2!)
- } 1 passo ricorsivo che contiene due chiamate ricorsive



Definizione ricorsiva della successione di Fibonacci

I numeri di Fibonacci

- Modello alla base di molte dinamiche evolutive delle popolazioni



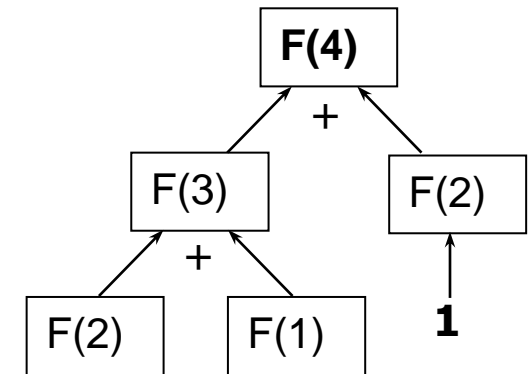
$$F = \{f_1, \dots, f_n\}$$

- $f_1 = 1$
 - $f_2 = 1$
 - Per $n > 2$, $f_n = f_{n-1} + f_{n-2}$
- } casi base (sono 2!)
- } 1 passo ricorsivo che contiene due chiamate ricorsive

Definizione ricorsiva della successione di Fibonacci

I numeri di Fibonacci

- Modello alla base di molte dinamiche evolutive delle popolazioni



$$F = \{f_1, \dots, f_n\}$$

- $f_1 = 1$
 - $f_2 = 1$
 - Per $n > 2$, $f_n = f_{n-1} + f_{n-2}$
- } casi base (sono 2!)
- } 1 passo ricorsivo che contiene due chiamate ricorsive



Definizione ricorsiva della successione di Fibonacci

I numeri di Fibonacci

- Modello alla base di molte dinamiche evolutive delle popolazioni

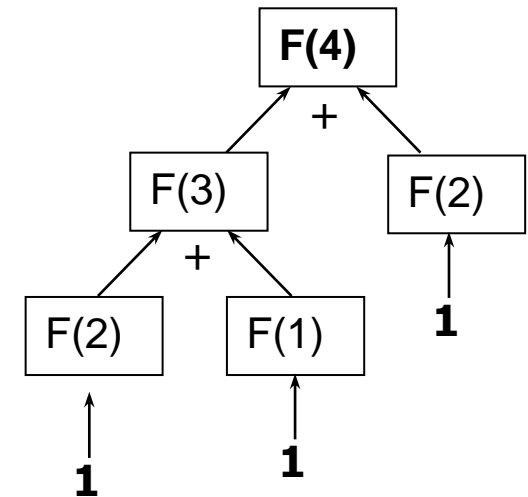
$$F = \{f_1, \dots, f_n\}$$

- $f_1 = 1$
- $f_2 = 1$

- Per $n > 2$, $f_n = f_{n-1} + f_{n-2}$

} casi base (sono 2!)

} 1 passo ricorsivo che
contiene due chiamate
ricorsive





Esempio: Fibonacci

È una sequenza di numeri interi in cui ogni numero si ottiene sommando i due precedenti nella sequenza. I primi due numeri della sequenza sono per definizione pari ad 1.

- $f_1 = 1$ (caso base)
- $f_2 = 1$ (caso base)
- $f_n = f_{n-1} + f_{n-2}$ (passo ricorsivo)



Esempio: Fibonacci

È una sequenza di numeri interi in cui ogni numero si ottiene sommando i due precedenti nella sequenza. I primi due numeri della sequenza sono per definizione pari ad 1.

- $f_1 = 1$ (caso base)
- $f_2 = 1$ (caso base)
- $f_n = f_{n-1} + f_{n-2}$ (passo ricorsivo)

```
int FiboRic(int n) {  
    if(n==1 || n==2) {  
        return 1;  
    }  
    else {  
        return FiboRic(n-2)+FiboRic(n-1);  
    }  
}
```



Problemi nell'uso della ricorsione

Uso della memoria

- La programmazione ricorsiva comporta spesso un uso inefficiente della memoria per la gestione degli spazi di lavoro delle chiamate generate
- In alcuni casi (che non abbiamo il tempo di vedere) viene comunque preferita ad altri approcci per la sua eleganza e semplicità
- In altri casi, si può ricorrere ad implementazioni iterative



Problemi nell'uso della ricorsione

Uso della memoria

- La programmazione ricorsiva comporta spesso un uso inefficiente della memoria per la gestione degli spazi di lavoro delle chiamate generate
- In alcuni casi (che non abbiamo il tempo di vedere) viene comunque preferita ad altri approcci per la sua eleganza e semplicità
- In altri casi, si può ricorrere ad implementazioni iterative

Funzione iterativa che calcola la somma dei primi n numeri della successione di fibonacci



Problemi nell'uso della ricorsione

Uso della memoria

- La programmazione ricorsiva comporta spesso un uso inefficiente della memoria per la gestione degli spazi di

```
int fibIter(int N) {  
    int i, N, n_1, n_2, fib;
```

- In (al tempo di vedere) viene accettato per la sua eleganza e

```
    n_1 = 1;
```

```
    n_2 = 1;
```

```
    fib = 1;
```

- In implementazioni iterative

```
    for(i = 3; i <= N; i++) {
```

```
        fib = n_1 + n_2;
```

```
        n_1 = n_2;
```

```
        n_2 = fib;
```

```
    }
```

```
    return fib;
```

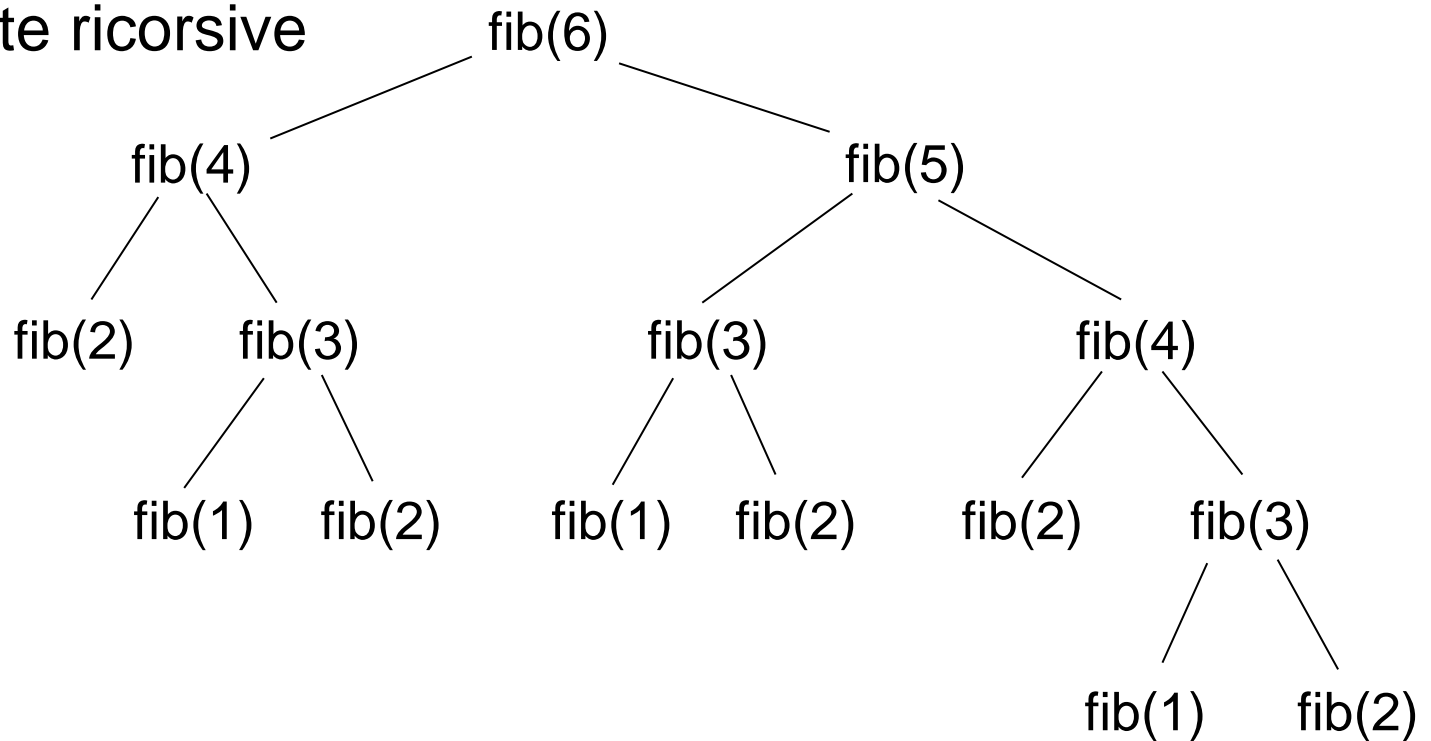
```
}
```

Funzione iterativa che calcola la somma dei primi n numeri della successione di fibonacci



Ricorsione Eccessiva

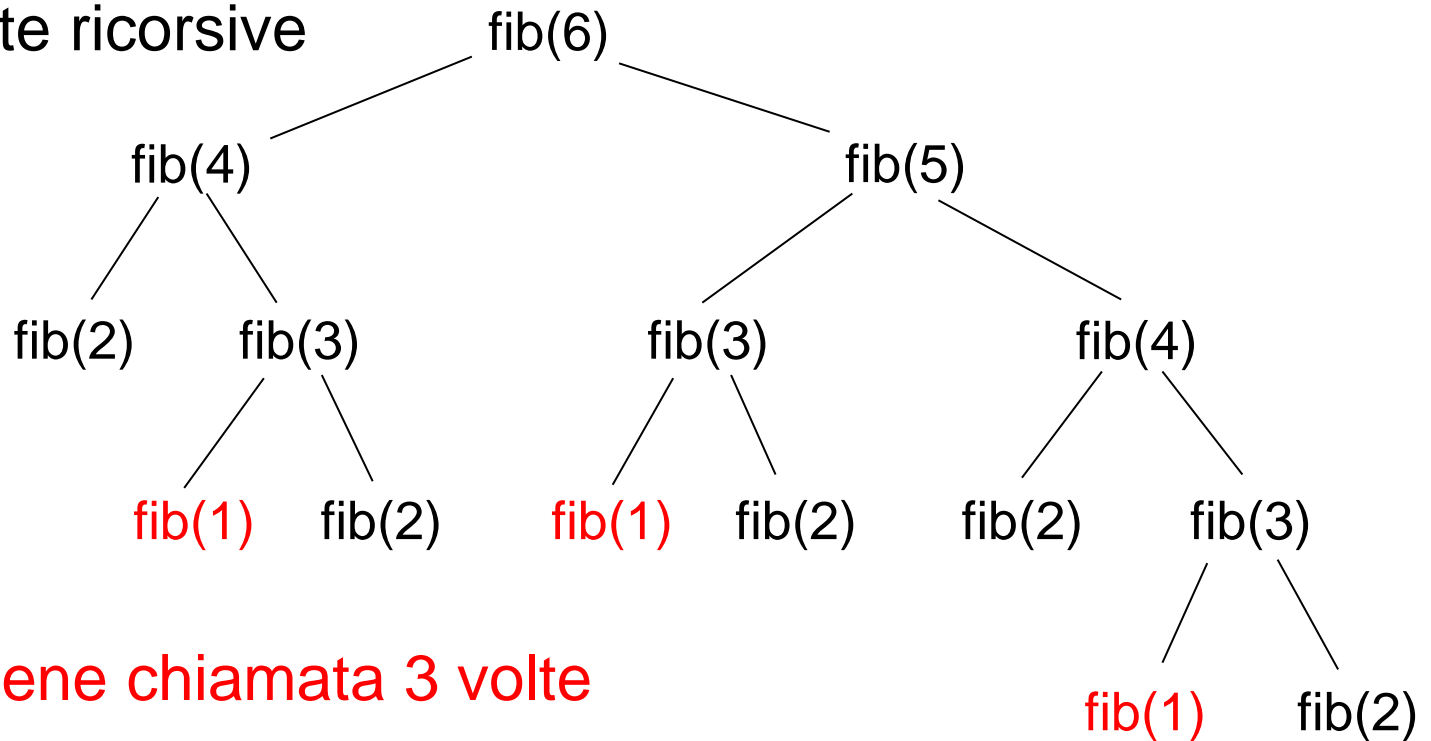
- Soluzione elegante ma dispendiosa: numero molto alto di chiamate ricorsive





Ricorsione Eccessiva

- Soluzione elegante ma dispendiosa: numero molto alto di chiamate ricorsive

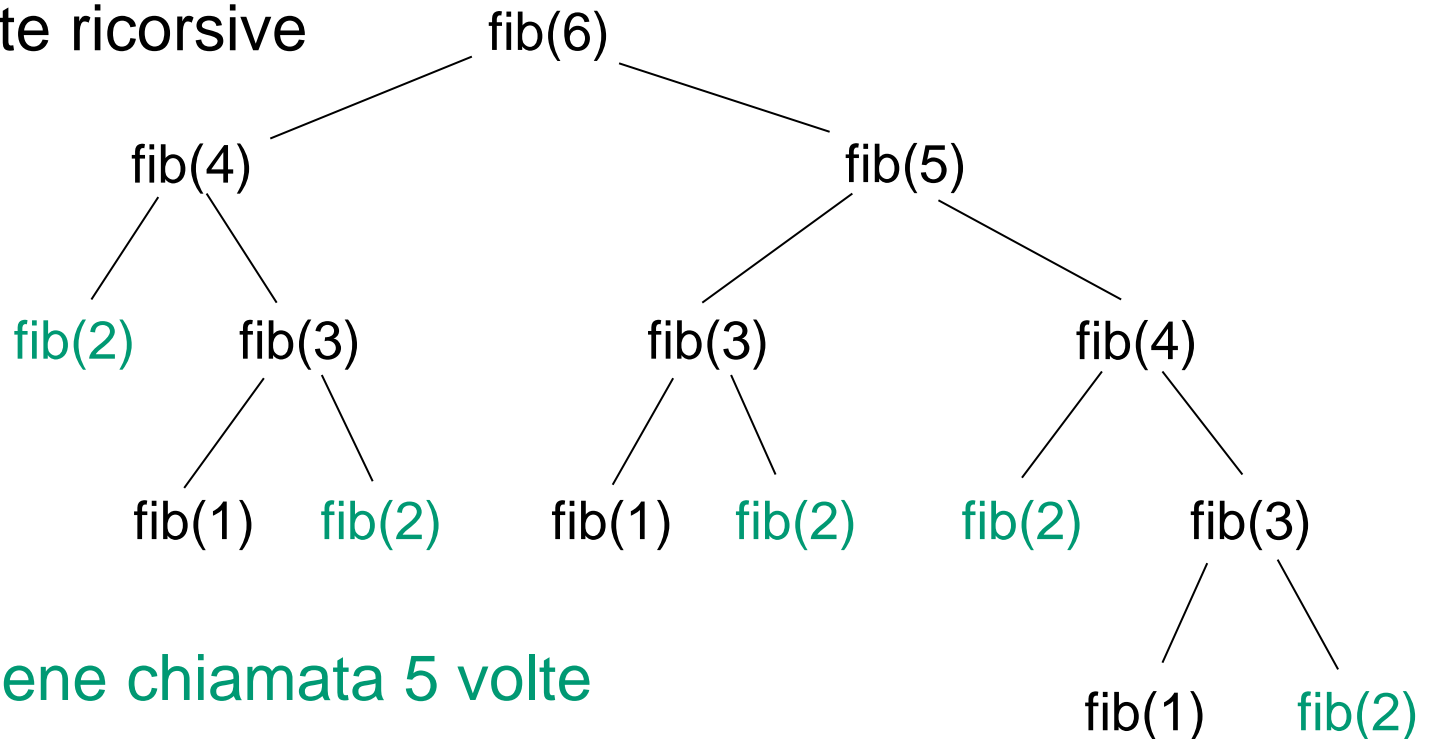


- $\text{fib}(1)$ viene chiamata 3 volte



Ricorsione Eccessiva

- Soluzione elegante ma dispendiosa: numero molto alto di chiamate ricorsive

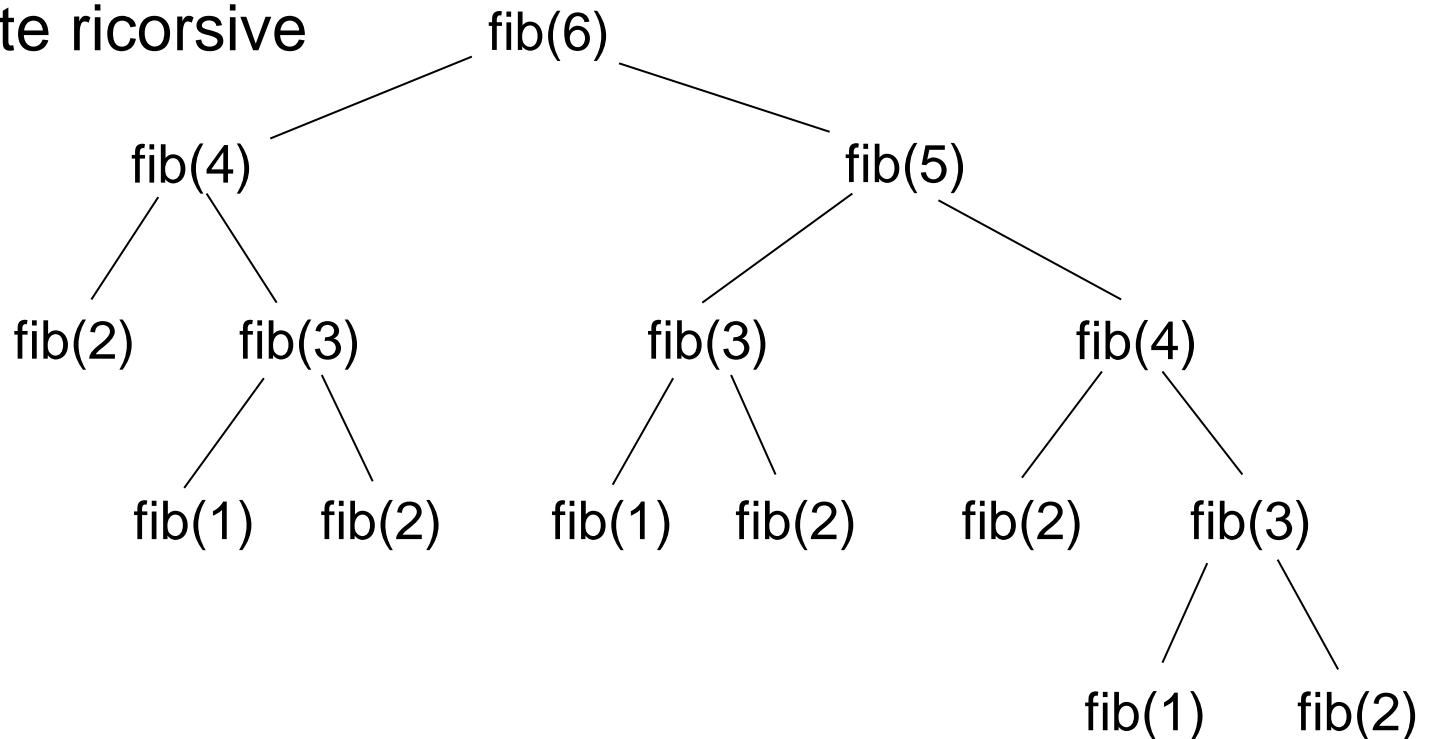


- $\text{fib}(2)$ viene chiamata 5 volte



Ricorsione Eccessiva

- Soluzione elegante ma dispendiosa: numero molto alto di chiamate ricorsive



- Provate a far calcolare fib(10), fib(15), fib(20), fib(25), fib(30)
- Quante volte viene calcolato fib(3)?
- Meglio usare una soluzione non ricorsiva...