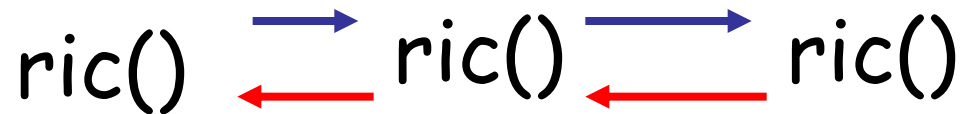


liste

concatenate

ricordare che nella ricorsione:

andata



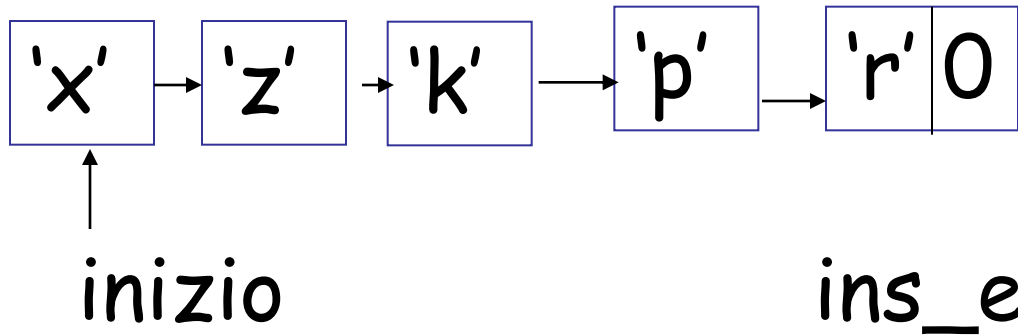
ritorno

calcolare all'andata e/o al ritorno

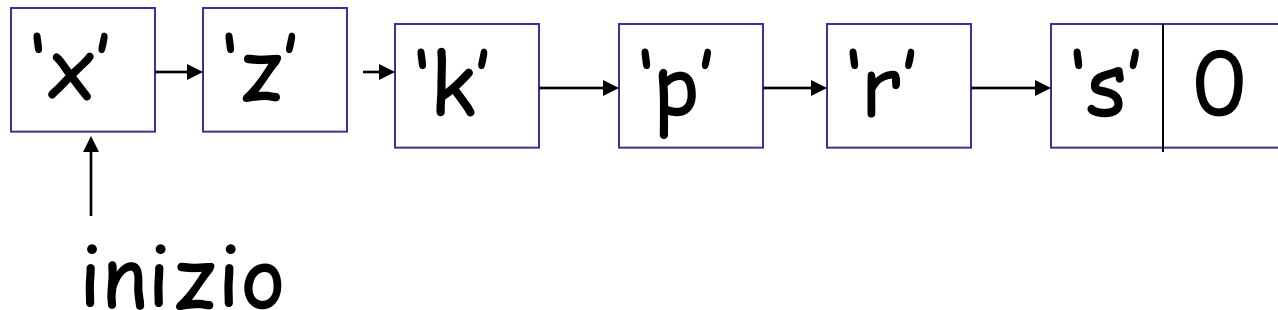
se non si fa nulla al ritorno allora ricorsione  
**terminale** → facile trasformarla in WHILE

```
struct nodo{char info; nodo* next;  
nodo(char a='\0', nodo*b=0){info=a;  
next=b;} };
```

# inserire un elemento alla fine della lista



`ins_end(inizio, 's')`

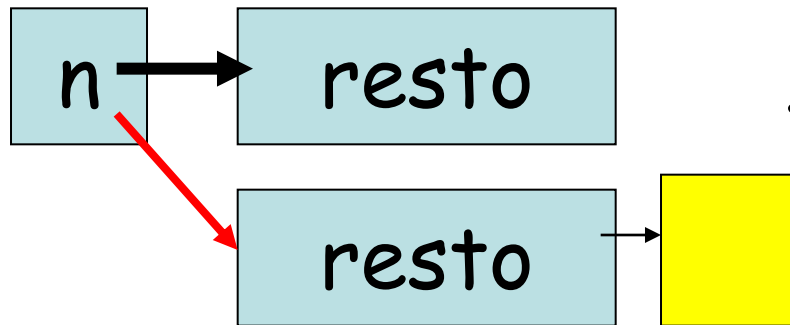


## I soluzione

**caso base:** lista vuota

*si tratta di creare il nuovo nodo e restituirlo*

**caso ricorsivo:** lista con un nodo almeno

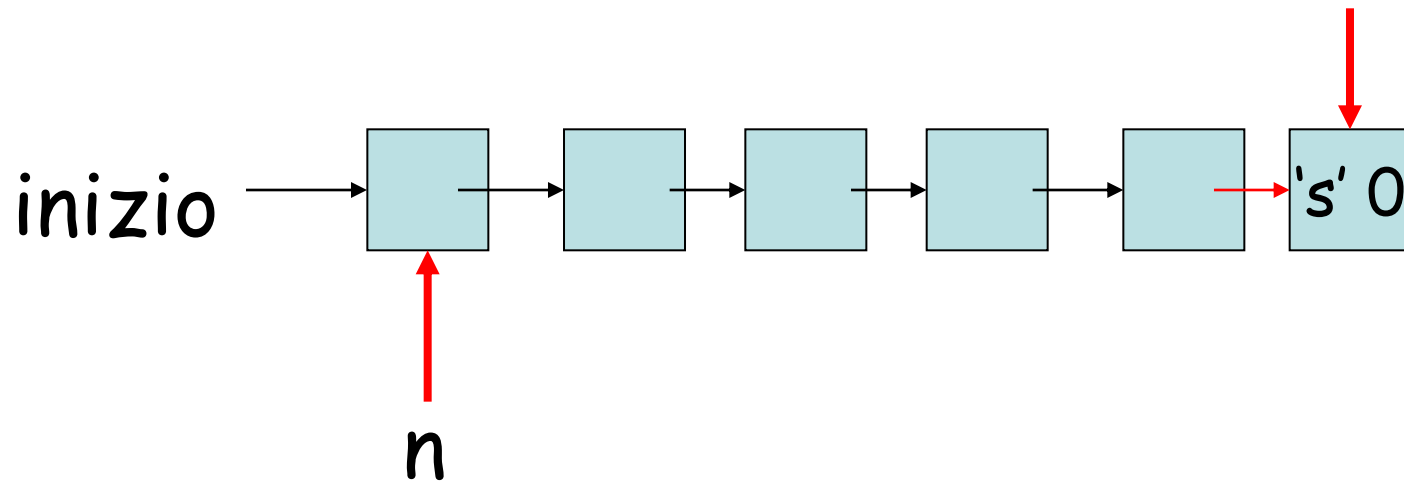


soluzione del resto

## Realizzazione:

.all'andata troviamo la fine della lista

.al ritorno costruiamo la lista allungata collegando ogni nodo con il nuovo resto



nella funzione che stiamo per  
descrivere

usiamo il fatto che :

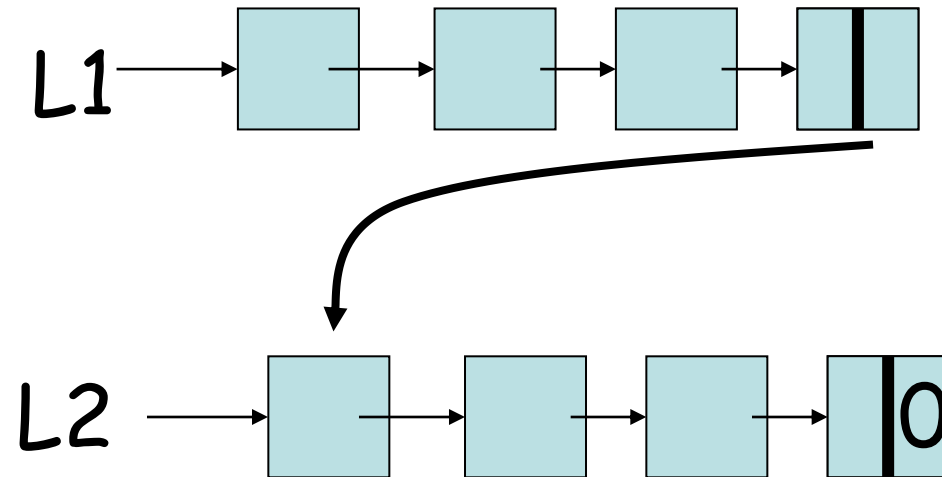
puntatore == 0 = false

puntatore != 0 = true

2 operazioni sulle liste:

- e::Resto esplicita il primo elemento  
oppure aggiunge e in cima a Resto
- L1 @ L2 concatena 2 liste

## L1 @ L2 concatena 2 liste





PRE=(lista(n) è lista corretta, event. vuota)

```
nodo * ins_end(nodo *n, char c)
{ if(! n) return new nodo(c,0);
  else
  {n→next=ins_end(n→next,c); return n; }
}
```

POST=(restituisce lista(n)@nodo(c,0))

## prova induttiva

**base**=lista(n) è 0 lista(n)@nodo(c,0) è nodo(c,0)

**passo ric**: lista(n)=a::lista(n'),


PRE\_ric vale  $\Rightarrow$  ins\_end(n $\rightarrow$ next,c)  
restituisce lista(n')@nodo(c,0)

restituiamo

a::lista(n')@nodo(c,0)=lista(n)@nodo(c,0)  
 $\Rightarrow$  POST

```
nodo * ins_end(nodo *n, char c)
{ if(! n)
  return new nodo(c,0);
else
  {n→next=ins_end(n→next,c); return n; }
}
```

fine  
ricorsione

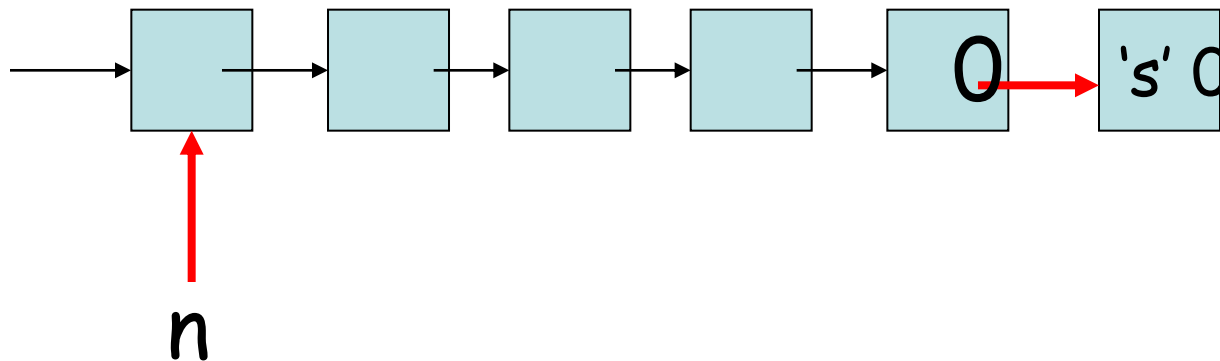


invocazione:  
**inizio**=ins\_end(inizio,'s');

invocazioni  
intermedie



**II soluzione:** facciamo tutto all'andata,  
significa che la funzione restituisce void  
e quindi non possiamo modificare **inizio**  
che punta al primo nodo  
funziona solo per liste non vuote



PRE=(lista(n) non vuota)

```
void ins_end(nodo *n, char c)
{ if( ! n→next)
    n→next=new nodo(c,0);
else
    ins_end(n→next,c);
}
```

POST=(restituisce lista(n)@nodo(c,0))

```
void ins_end(nodo *n, char c)
{ if( ! n→next)
    n→next=new nodo(c,0);
else
    ins_end(n→next,c);
}
```

da chiamare  
solo con  $n \neq 0$

```
if(inizio) ins_end(inizio, 's');
else inizio=new nodo('s',0);
```

**base:** lista(n) consiste di un solo nodo, la funzione costruisce lista(n)@nodo(c,0)

**passo ric:** lista(n)=a::a'::lista(n')

PRE\_ric vale, cioè a'::lista(n') non è vuota

vale POST\_ric, cioè

ins\_end(n→next,c) costruisce

a'::lista(n')@nodo(c,0)

e quindi adesso abbiamo

a::a'::lista(n')@nodo(c,0)

riassumiamo:

**soluzione I** : all'andata si passa l'ultimo nodo ( $n == 0$ ) e poi si costruisce la nuova lista al ritorno

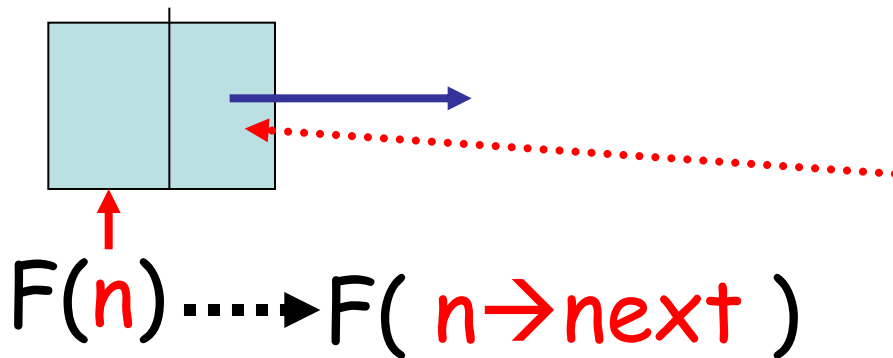
**soluzione II**: all'andata ci si ferma all'ultimo nodo ( $n \rightarrow next == 0$ ) e gli si attacca il nuovo nodo. Non si fa nulla al ritorno

la II è più semplice, ma non gestisce il caso della lista vuota



vorremmo contemporaneamente  
poter modificare il campo next  
dell'ultimo nodo ma fermarci con  $n == 0$   
possiamo ottenerlo passando il nodo  
per riferimento

$F(\text{nodo} * \& n) \{ \dots F(n \rightarrow \text{next}) \dots \}$



l'invocazione ha  
un alias di questo  
campo di  $n$

### III soluzione: con pass. per riferimento

PRE=(lista(n) è lista corretta = vL)

```
void ins(nodo*&n,int x)
```

```
{
```

```
  if(!n)
```

```
    n=new nodo(x,0);
```

```
  else
```

```
    ins(n->next,x);
```

```
} POST=(lista(n) = vL@nodo(x,0))
```

**base:**  $n=0 \Rightarrow vL$  è vuota,  
 $lista(n)=vL@nodo(c,0)=nodo(c,0)$

**passo ric:**  $vL=a::vL'$

$n \rightarrow next$  punta a lista  $vL' \Rightarrow$  vale  $PRE\_ric$

$lista(n \rightarrow next)=vL'@nodo(c,0)$

$lista(n)=a::vL'@nodo(c,0)=vL@nodo(c,0)$

$\Rightarrow$

POST