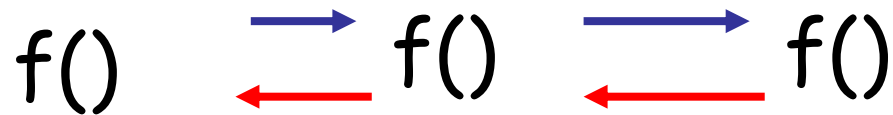


liste

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;} };
```

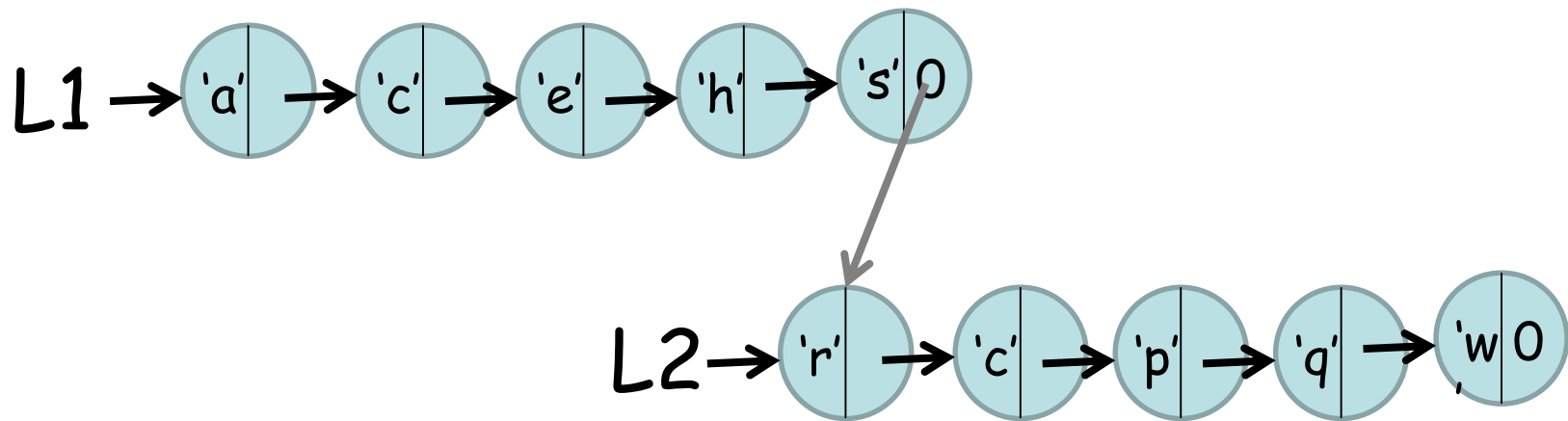
una lista è una variabile `nodo*` che punta al suo primo nodo (oppure 0/NULL)

usiamo `n`, `L`, `L1`, `L2`....; e spesso confonderemo `n` ed `L` con il nodo puntato, `*n` e `*L`

notazione: `L(n)`, `L(L)` per lista puntata da `n` e `L` è **corretta** se è 0 o punta ad un nodo che punta ad una lista corretta (resto della lista)

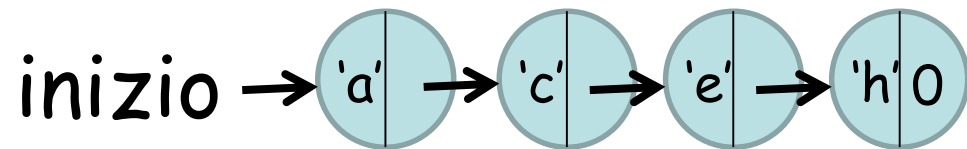
per parlare di liste ci serve notazione
per le liste:

-L1 @ L2 concatena 2 liste

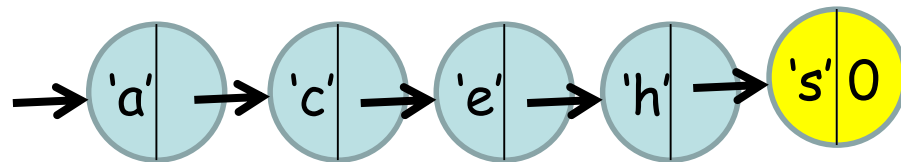


potrebbe essere che L1 ed L2 siano
vuote o contengano solo 1 nodo

Problema: inserire un elemento alla fine
di una lista lista



`ins_end(inizio, 's')` restituisce



I soluzione

diamo la lista iniziale in input e
otteniamo la **nuova** lista come risultato

nodo* ins_end(nodo*,char)

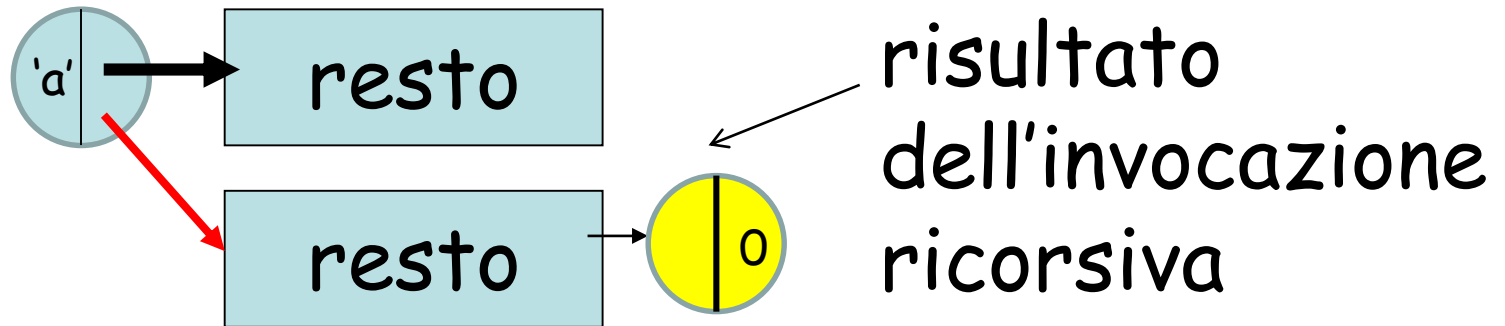
tutto passato per valore

caso base: lista vuota

`return new nodo('s',0);`

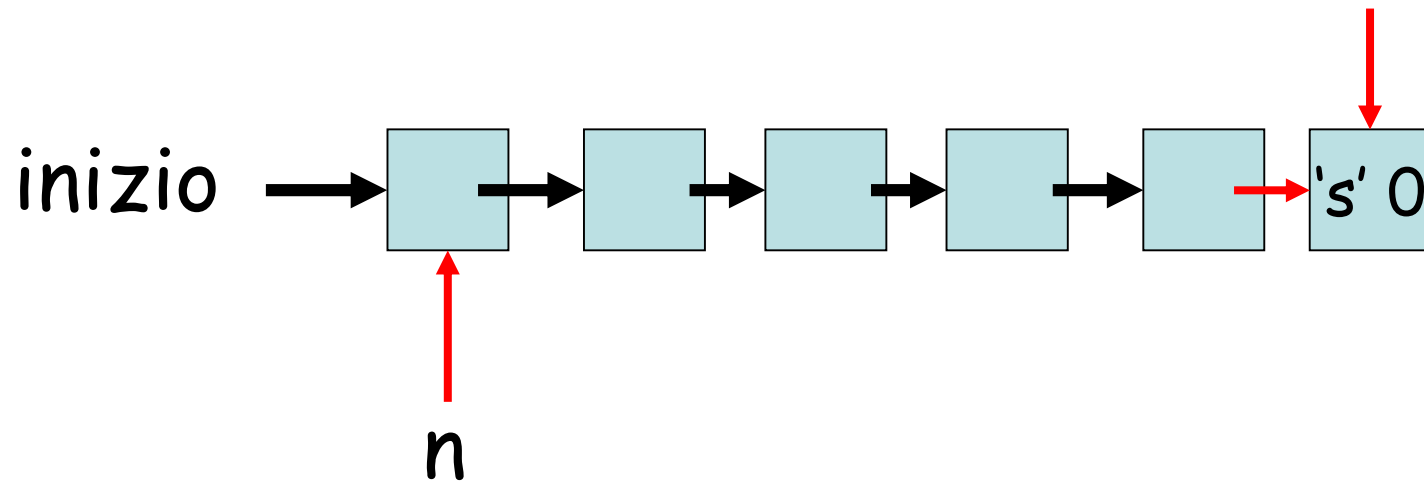
caso ricorsivo: lista con un nodo almeno

invocazione ricorsiva
sul resto



Realizzazione:

- all'andata della ricorsione troviamo la fine della lista
- al ritorno costruiamo la lista allungata collegando ogni nodo con il nuovo resto

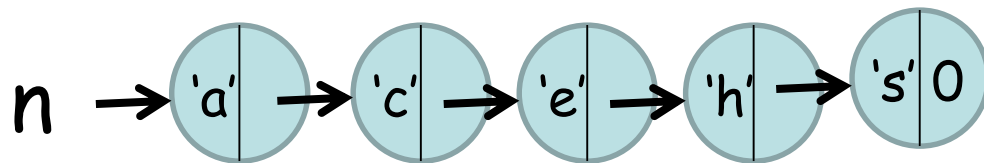


nella funzione che stiamo per descrivere

usiamo il fatto che :

puntatore == 0 = false

puntatore != 0 = true



ricordare: $L(n)$ indica l'intera lista

PRE=(L(n) è lista corretta, vL(n)=L(n))

```
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 vL(n)@nodo(c,0))

prova induttiva

base= $vL(n)=0$, $vL(n)@nodo(c,0)=nodo(c,0)$

passo induttivo: $vL(n)=n@vL(n \rightarrow next)$,

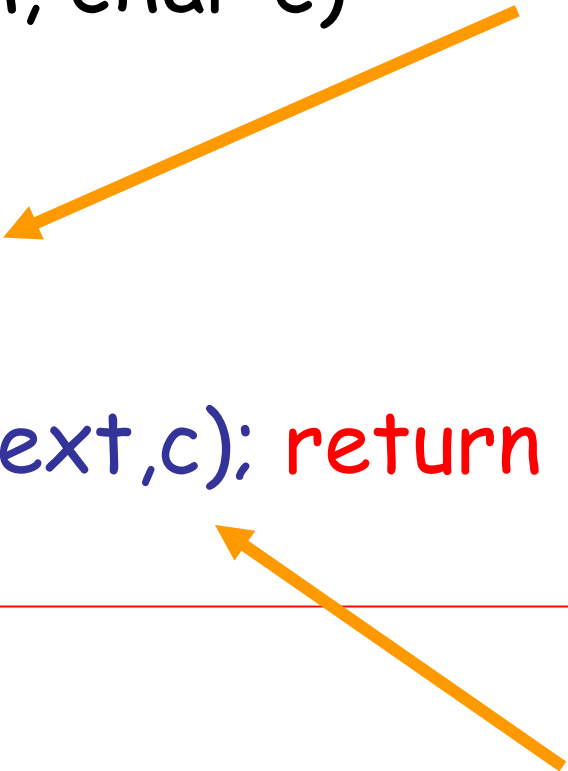
$PRE_ric=(vL(n \rightarrow next))$ è corretta, vale \Rightarrow
vale $POST_ric$ e cioè $ins_end(n \rightarrow next, c)$
restituisce $vL(n \rightarrow next) @ nodo(c,0)$

la funzione restituisce

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

$vL(n) @ nodo(c,0) \Rightarrow POST$

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



caso base

invocazioni
ricorsive

invocazione iniziale:

inizio=ins_end(inizio,'s');

insomma:

le operazioni che fa sono:

- scorrere la lista fino a lista vuota=0

- la creazione del nuovo nodo

- suo aggancio a quello che era l'ultimo nodo, **se c'era**

- restituzione dei suffissi aumentati di un nodo e loro aggancio

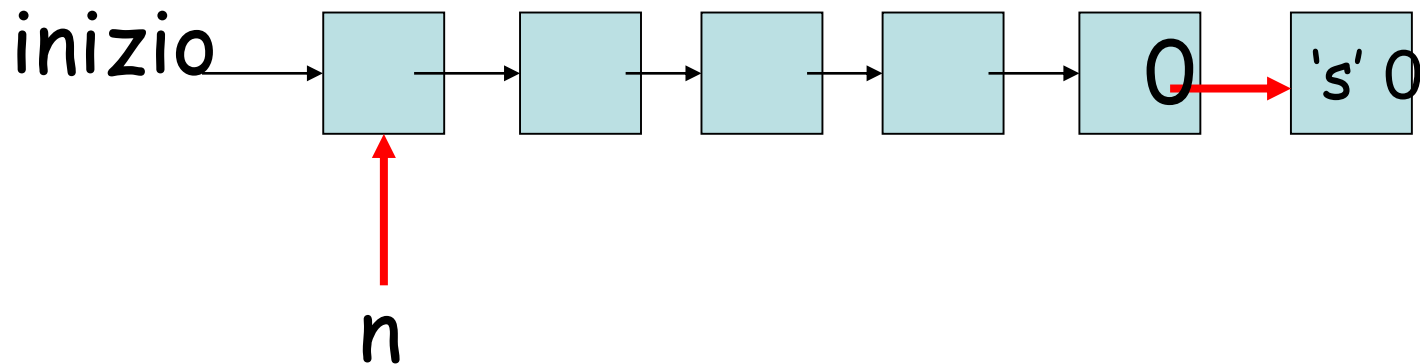
- la prima invocazione restituisce la lista originale con il nodo in più alla fine

le cose grigie sono inutili !!

II soluzione: all'andata ci fermiamo all'ultimo nodo e gli appendiamo il nuovo nodo

....ma ci deve essere **almeno un nodo**

➔ inizio non può cambiare, ritorna void



PRE=(L(n) è corretta e non vuota vL(n)=L(n))

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

POST=(L(n)=vL(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: $vL(n)=n$, la funzione restituisce
 $n @ \text{nodo}(c,0) \Rightarrow \text{POST}$

passo ric:

$vL(n)=n @ (n \rightarrow \text{next}) @ vL(n \rightarrow \text{next} \rightarrow \text{next})$

$\text{PRE_ric vale}=(n \rightarrow \text{next}) @ vL(n \rightarrow \text{next} \rightarrow \text{next})$ è
corretta e non è vuota

vale POST_ric , cioè dopo $\text{ins_end}(n \rightarrow \text{next}, c)$

$L(n \rightarrow \text{next})=vL(n \rightarrow \text{next}) @ \text{nodo}(c,0)$

e quindi adesso abbiamo

$L(n)=n @ vL(n \rightarrow \text{next}) @ \text{nodo}(c,0) = vL(n) @ \text{nodo}(c,0)$

riassumiamo :

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

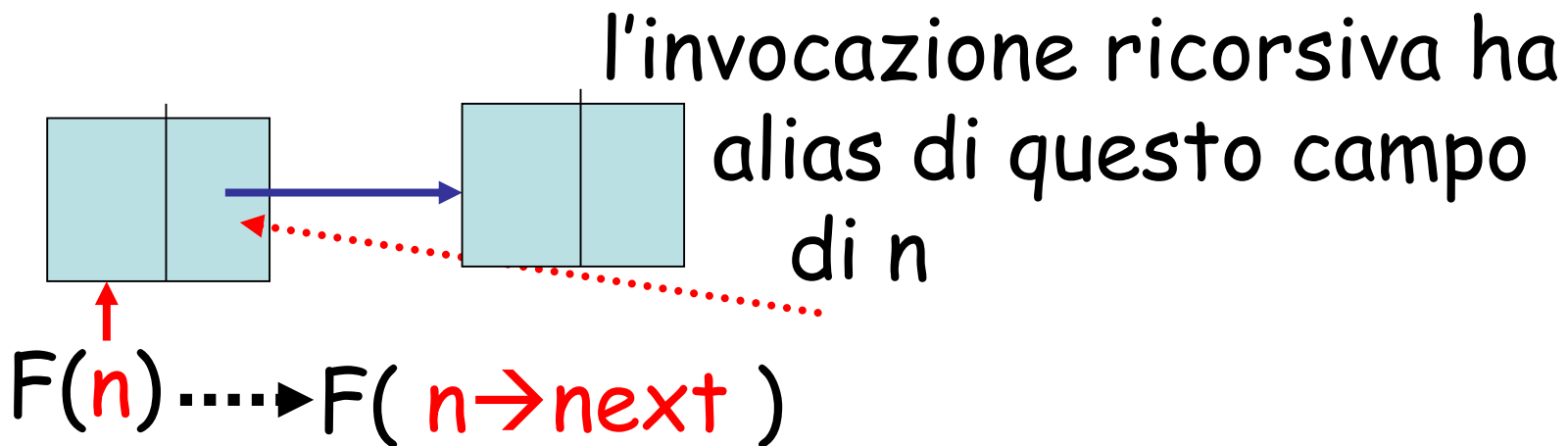
soluzione II: all'andata ci si ferma all'ultimo nodo (caso base $n \rightarrow \text{next} == 0$) e gli si attacca il nuovo nodo. Non si fa nulla al ritorno

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

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



III soluzione: con pass. per riferimento

PRE=(L(n) è lista corretta, vL(n)=L(n))

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

```
{
```

```
  if(!n)
```

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

```
  else
```

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

```
}
```

POST=(L(n) = vL(n) @ nodo(c,0))

dimostrazione:

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

passo ric: $vL(n)=n @ vL(n \rightarrow next)$
 $vL(n \rightarrow next)$ è lista corretta \Rightarrow vale PRE_ric
POST_ric \Rightarrow
 $L(n \rightarrow next)=vL(n \rightarrow next) @ nodo(c,0)$
 $L(n)=n @ L(n \rightarrow next)=vL(n) @ nodo(c,0)$
 \Rightarrow
POST = $(L(n)=vL(n) @ nodo(c,0))$