Lezione 26 – ricorsione su liste

- interpretazione *induttiva* di una lista
  - una *lista* e` un *elemento* seguito da una *lista* di elementi
  - non proprio, ma quasi

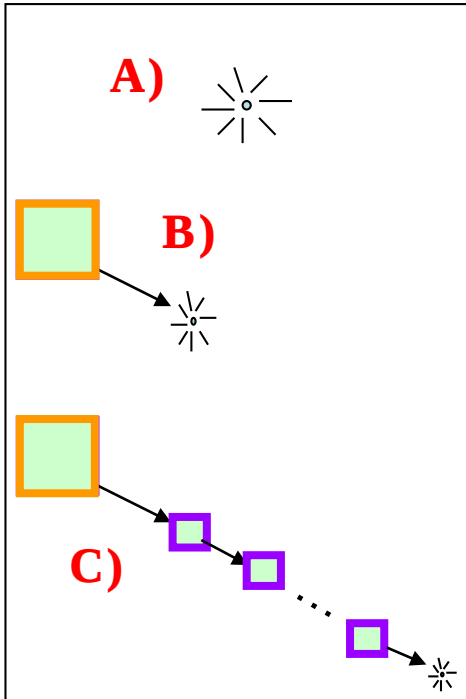
- vari esercizi

LISTE - interpretazione induttiva - esempio

Possiamo interpretare una sequenza di elementi come un **elemento** seguito da una **(sotto)sequenza che a sua volta e` una sequenza**

lista(*non vuota*) = 1ºelemento + restolista

(ma solo se la sequenza e` non vuota). Ampliando questa osservazione, si distinguono tre casi (A, B, C) che convergono nella definizione induttiva qui sotto



A) vuota

B) un solo elem. (+lista resto vuota)

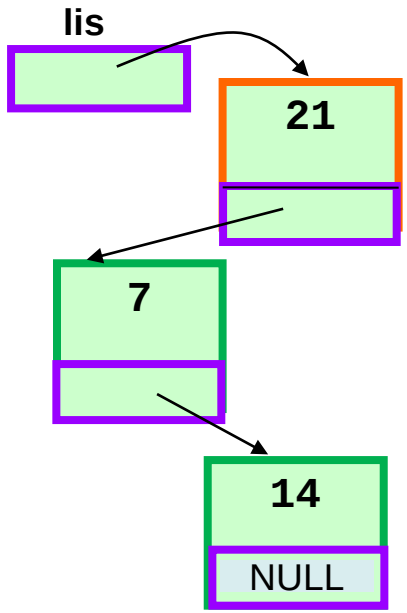
C) un elem. + lista resto

tutti questi casi si possono raccogliere nella definizione:

una lista e` un insieme di elementi che

- o e` vuoto
- o e` scindibile in (primo) elemento e lista degli elementi rimanenti

$$\text{LISTA} = \begin{cases} \text{LISTA VUOTA} \\ \text{elemento} + \text{restolista} \quad (\text{LISTA}) \end{cases}$$

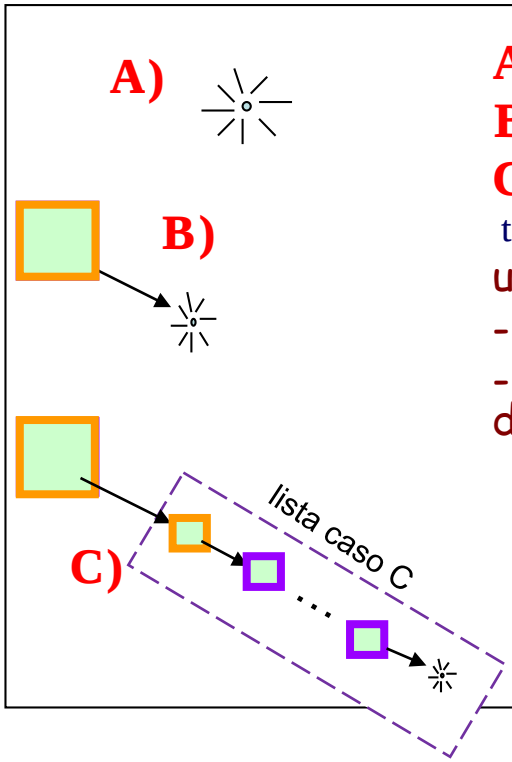


LISTE - interpretazione induttiva - esempio

Possiamo interpretare una sequenza di elementi come un **elemento** seguito da una **(sotto)sequenza che a sua volta e` una sequenza**

lista(*non vuota*) = 1ºelemento + restolista

(ma solo se la sequenza e` non vuota). Ampliando questa osservazione, si distinguono tre casi (A, B, C) che convergono nella definizione induttiva qui sotto



A) vuota

B) un solo elem. (+lista resto vuota)

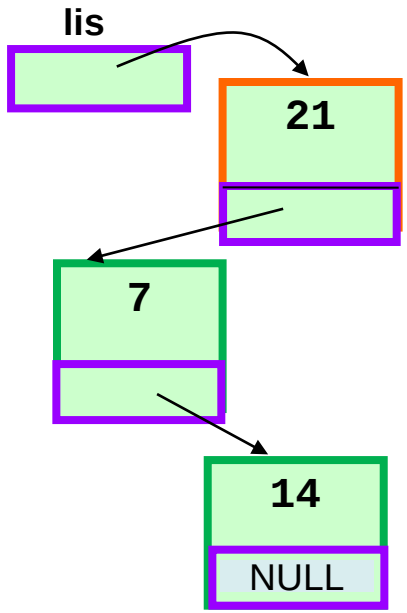
C) un elem. + lista resto

tutti questi casi si possono raccogliere nella definizione:

una lista e` un insieme di elementi che

- o e` vuoto
- o e` scindibile in (primo) elemento e lista degli elementi rimanenti

$$\text{LISTA} = \begin{cases} \text{LISTA VUOTA} \\ \text{elemento} + \text{restolista} \quad (\text{LISTA}) \end{cases}$$

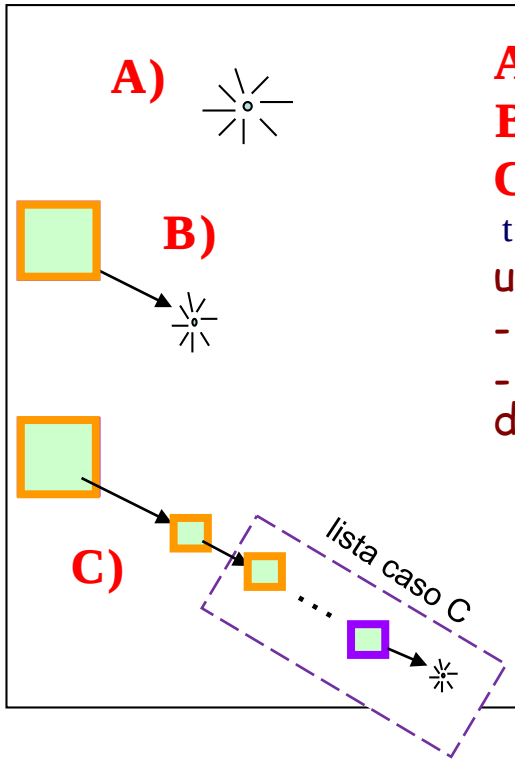


LISTE - interpretazione induttiva - esempio

Possiamo interpretare una sequenza di elementi come un **elemento** seguito da una **(sotto)sequenza che a sua volta e` una sequenza**

lista(*non vuota*) = 1ºelemento + restolista

(ma solo se la sequenza e` non vuota). Ampliando questa osservazione, si distinguono tre casi (A, B, C) che convergono nella definizione induttiva qui sotto



A) vuota

B) un solo elem. (+lista resto vuota)

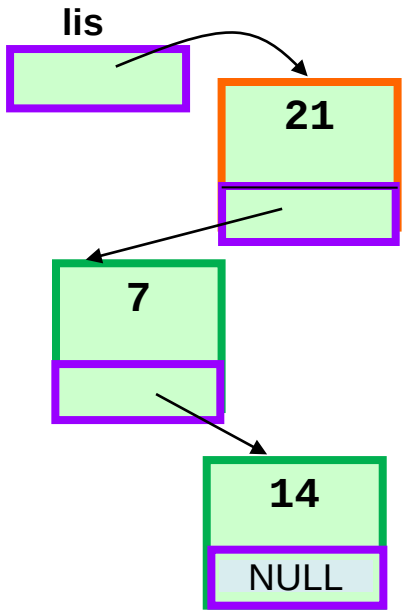
C) un elem. + lista resto

tutti questi casi si possono raccogliere nella definizione:

una lista e` un insieme di elementi che

- o e` vuoto
- o e` scindibile in (primo) elemento e lista degli elementi rimanenti

$$\text{LISTA} = \begin{cases} \text{LISTA VUOTA} \\ \text{elemento} + \text{restolista} \quad (\text{LISTA}) \end{cases}$$

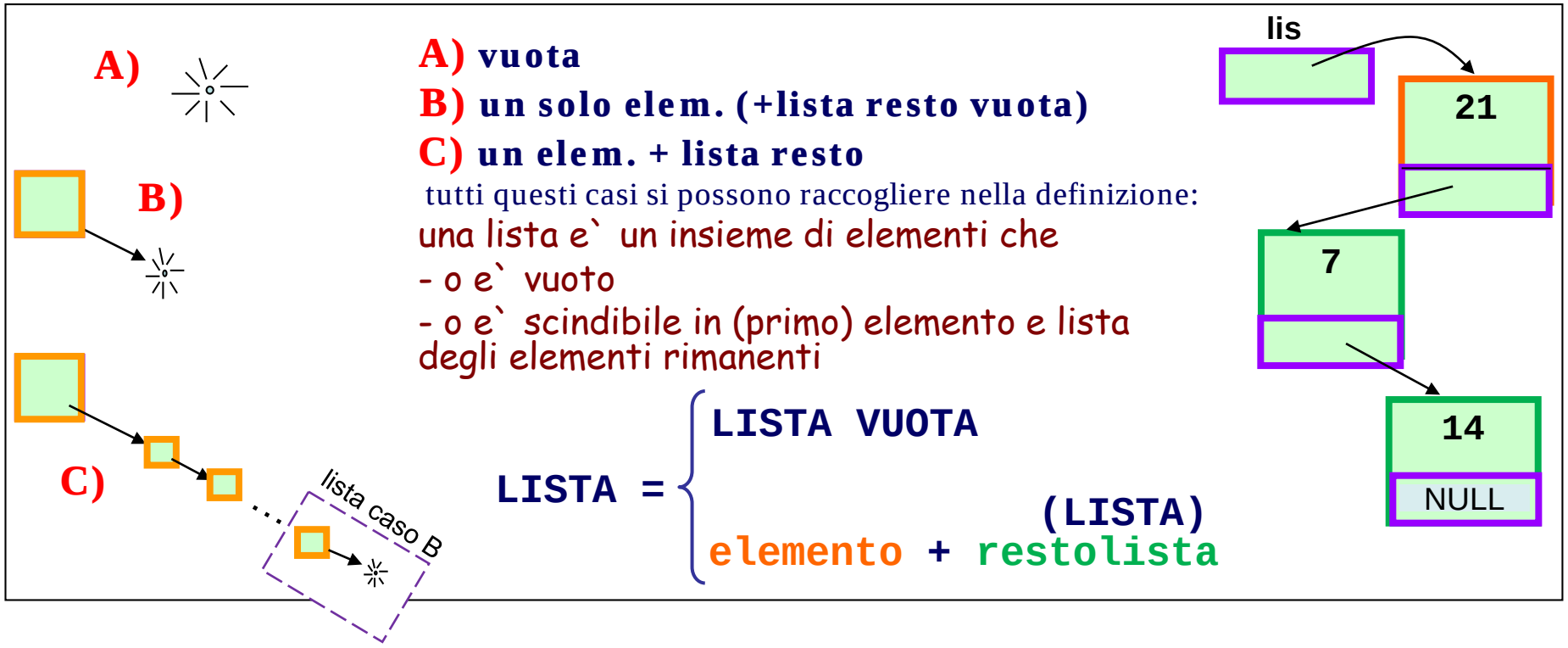


LISTE - interpretazione induttiva - esempio

Possiamo interpretare una sequenza di elementi come un **elemento** seguito da una **(sotto)sequenza che a sua volta e` una sequenza**

lista(*non vuota*) = 1ºelemento + restolista

(ma solo se la sequenza e` non vuota). Ampliando questa osservazione, si distinguono tre casi (A, B, C) che convergono nella definizione induttiva qui sotto

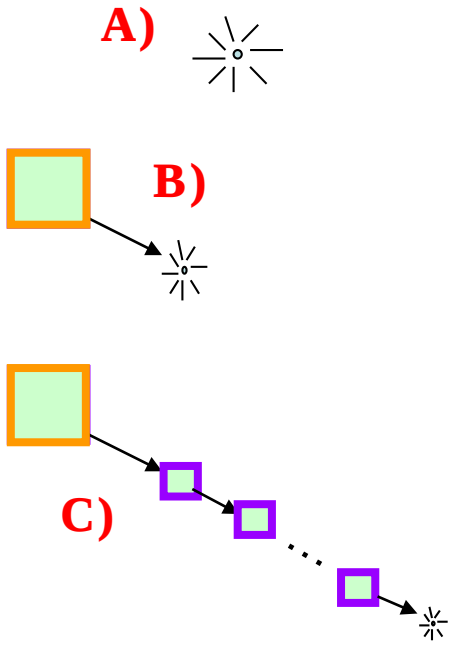


LISTE - interpretazione induttiva - esempio

Possiamo interpretare una sequenza di elementi come un **elemento** seguito da una (**sotto)sequenza che a sua volta e` una sequenza**

lista(*non vuota*) = 1ºelemento + restolista

(ma solo se la sequenza e` non vuota). Ampliando questa osservazione, si distinguono tre casi (A, B, C) che convergono nella definizione induttiva qui sotto



A) vuota

B) un solo elem. (+lista resto vuota)

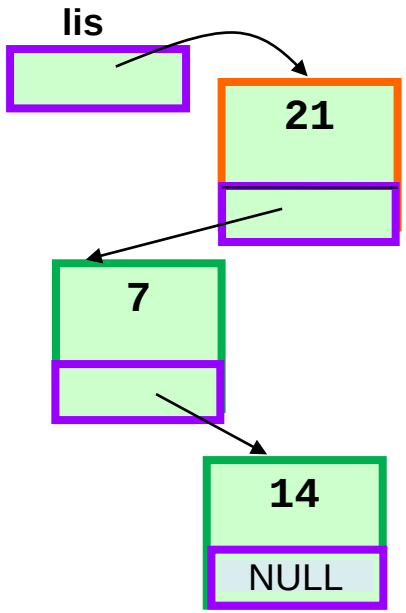
C) un elem. + lista resto

tutti questi casi si possono raccogliere nella definizione:

una lista e` un insieme di elementi che

- o e` vuoto
- o e` scindibile in (primo) elemento e lista degli elementi rimanenti

$$\text{LISTA} = \begin{cases} \text{LISTA VUOTA} \\ \text{elemento} + \text{restolista} \end{cases} \quad (\text{LISTA})$$

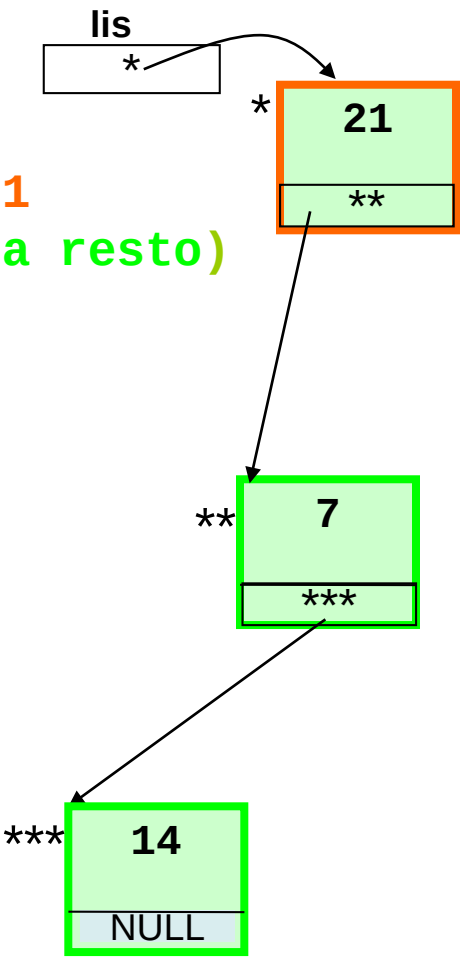


Per cui, l'espressione di una funzione F_{OP} che esegue OP sugli elementi della lista puo` essere data cosi`

$$F_{OP}(\text{LISTA}) = \begin{cases} \text{se LISTA VUOTA} & \text{NULLA} \\ \text{senno`} & \text{- OP sul primo elemento} \\ & \text{- } F_{OP}(\text{restolista}) \end{cases}$$

esempio: aggiungere 1 ad ogni elemento della lista

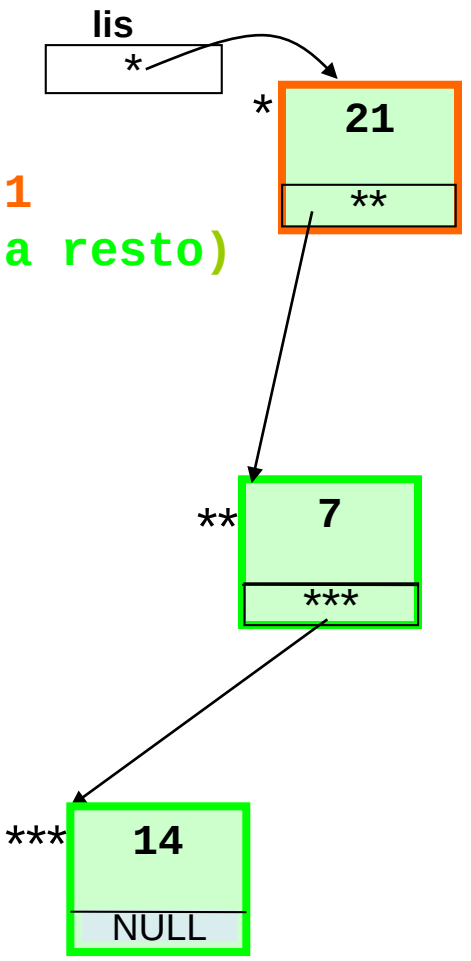
```
AGGIUNGI1(LISTA) = { se LISTA VUOTA      NULLA
                   { senno`              1) PRIMO.info += 1
                                         2) AGGIUNGI1(lista resto)
```




```
void aggiungiUno (     ☺     ) {

            ☺



return;
}
```

esempio: aggiungere 1 ad ogni elemento della lista

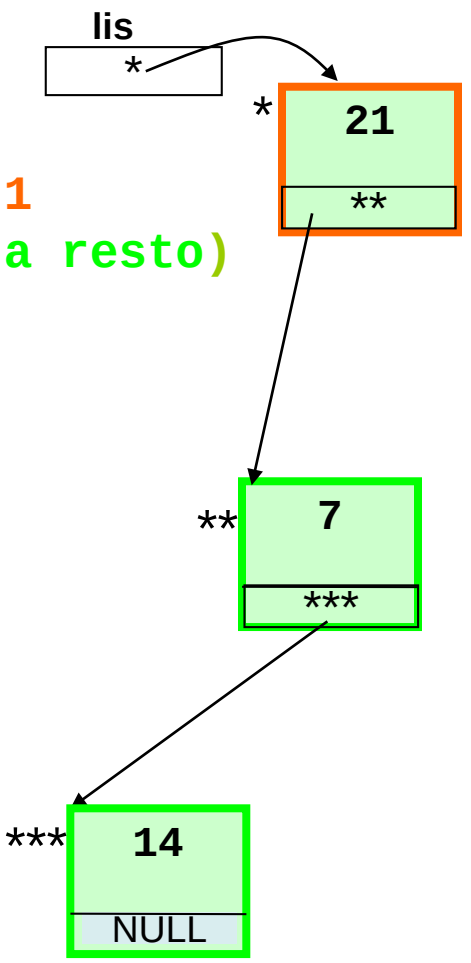
```
AGGIUNGI1(LISTA) = { se LISTA VUOTA        NULLA
                   { senno`                1) PRIMO.info += 1
                                           2) AGGIUNGI1(lista resto)
```




```
void aggiungiUno (TipoLista l) {
  if (l) {
    l->info += 1;
    aggiungiUno(l->next);
  }
return;
}
```

esempio: aggiungere 1 ad ogni elemento della lista

```
AGGIUNGI1(LISTA) = { se LISTA VUOTA      NULLA
                   { senno`              1) PRIMO.info += 1
                                         2) AGGIUNGI1(lista resto)
```




```
void aggiungiUno (TipoLista l) {
  if (l) {
    l->info += 1;
    aggiungiUno(l->next);
  }
return;
}
```

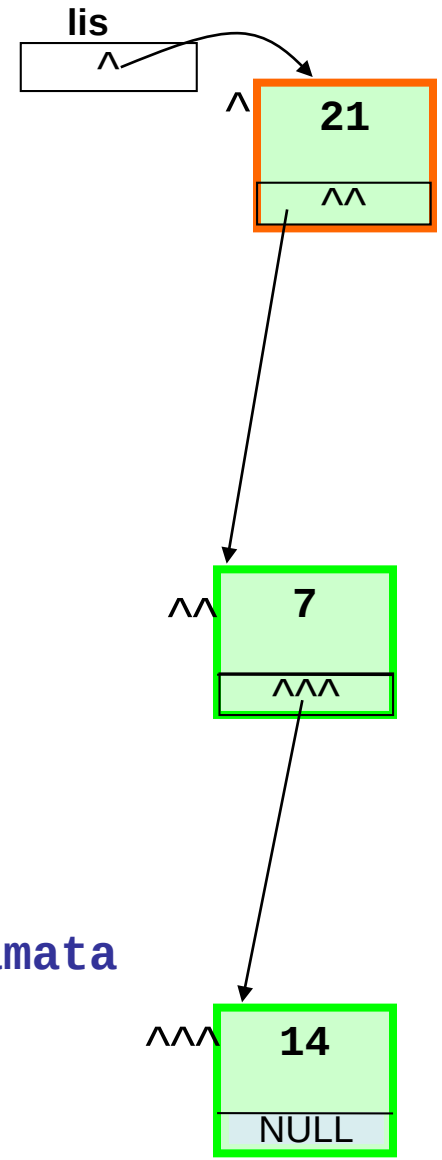
lista resto

PRIMO

LISTE - *interpretazione induttiva*

```
void aggiungiUno (TipoLista l) {
  if (l) {
    l->info += 1;
    aggiungiUno(l->next);
  }
return;
}
```

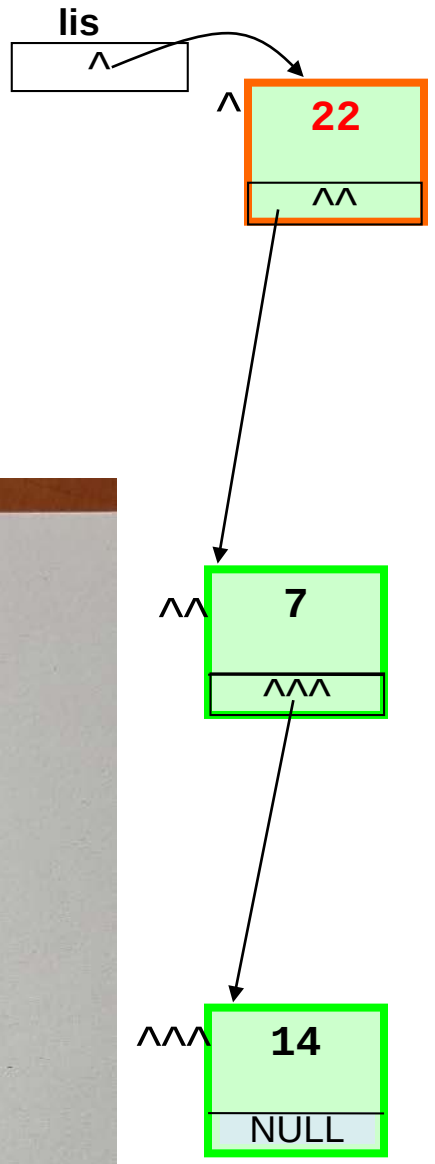
aggiungiUno(lis) **RDA**



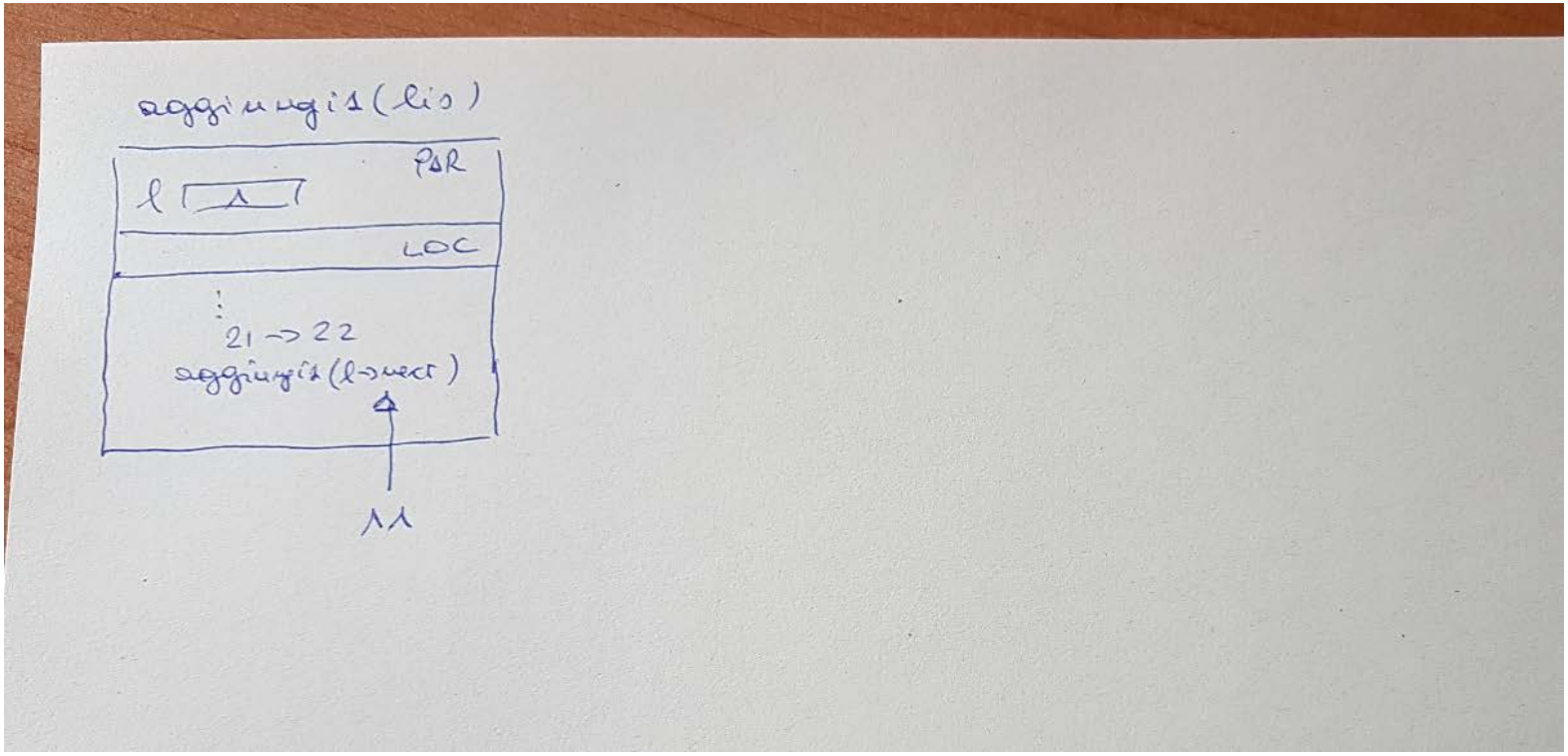
Disegnare i record di attivazione di questa chiamata

LISTE - *interpretazione induttiva*

```
void aggiungiUno (TipoLista l) {
  if (l) {
    l->info += 1;
    aggiungiUno(l->next);
  }
return;
}
```

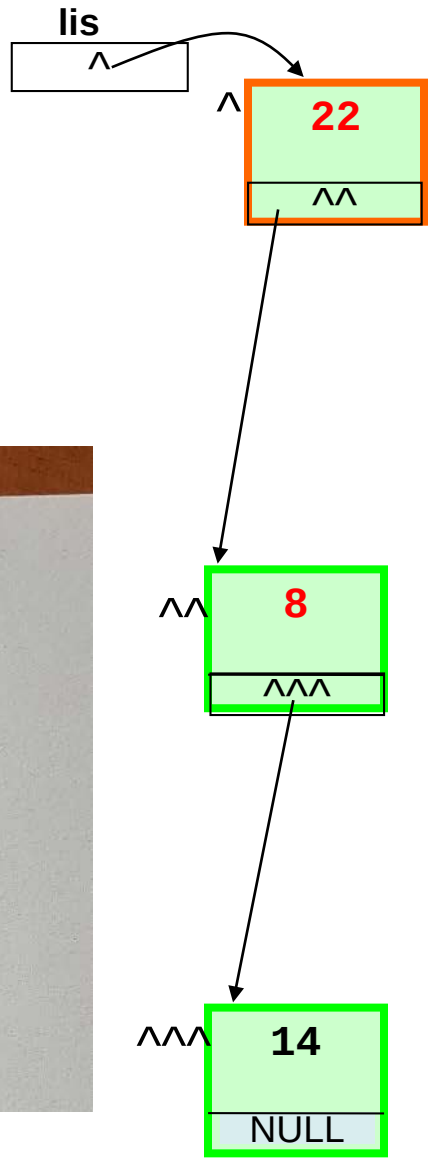


aggiungiUno(lis) **RDA**

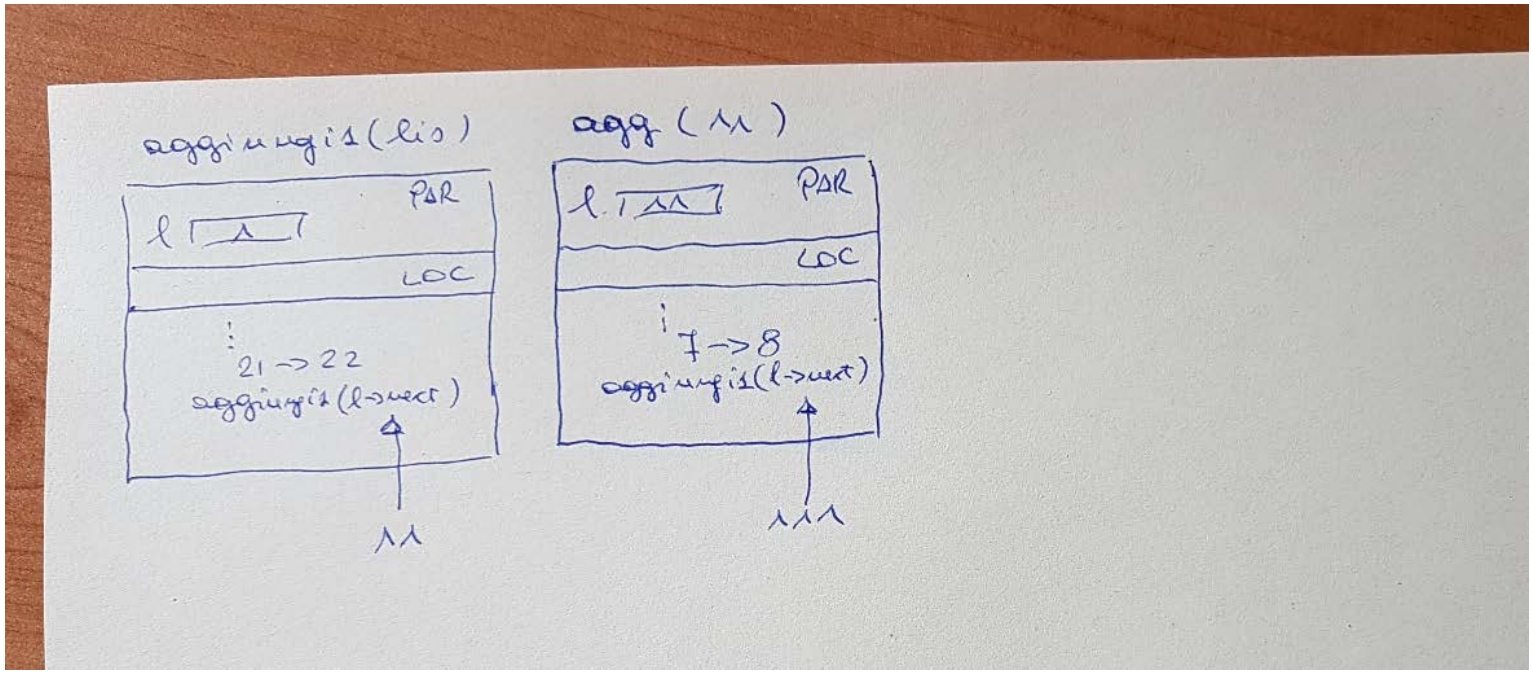


LISTE - *interpretazione induttiva*

```
void aggiungiUno (TipoLista l) {
  if (l) {
    l->info += 1;
    aggiungiUno(l->next);
  }
return;
}
```

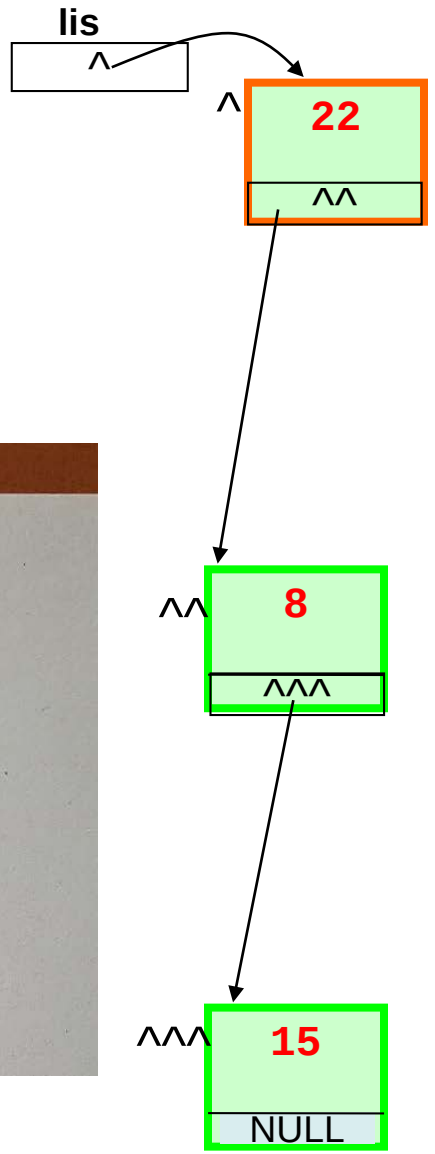


aggiungiUno(lis) **RDA**

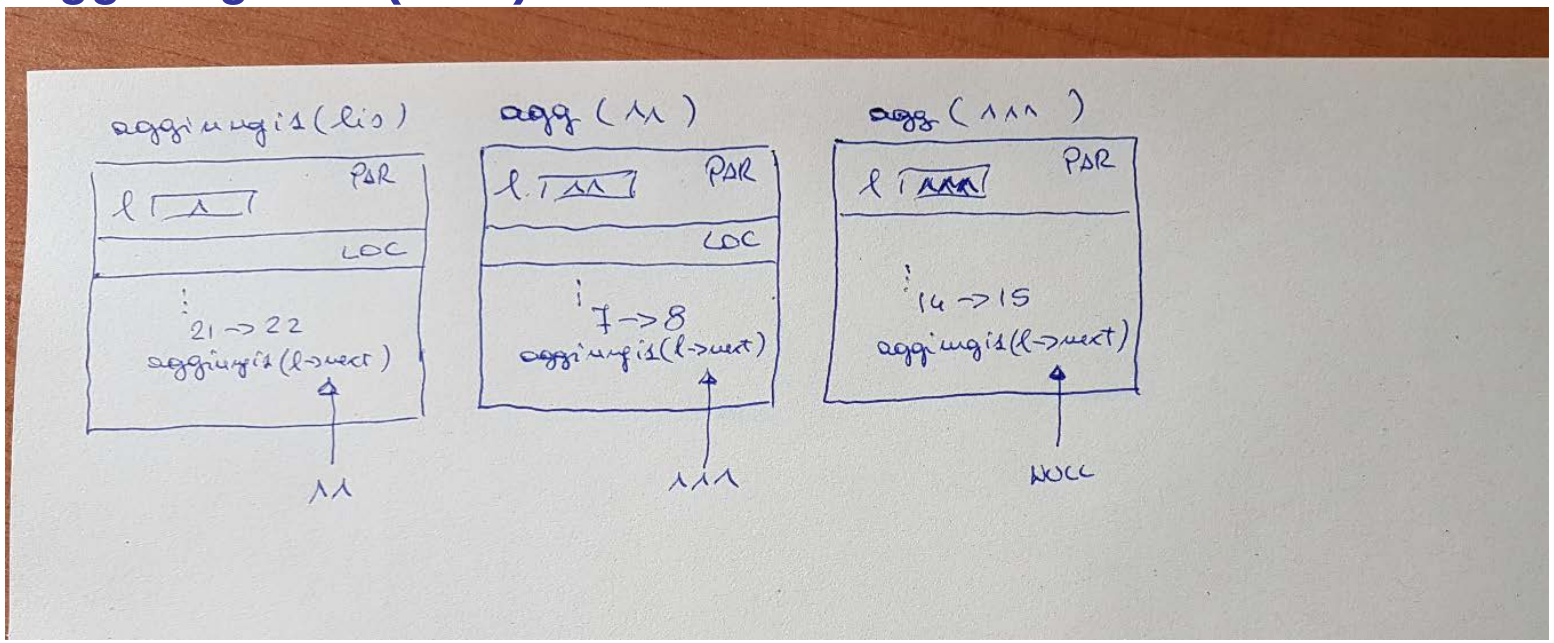


LISTE - *interpretazione induttiva*

```
void aggiungiUno (TipoLista l) {
  if (l) {
    l->info += 1;
    aggiungiUno(l->next);
  }
return;
}
```

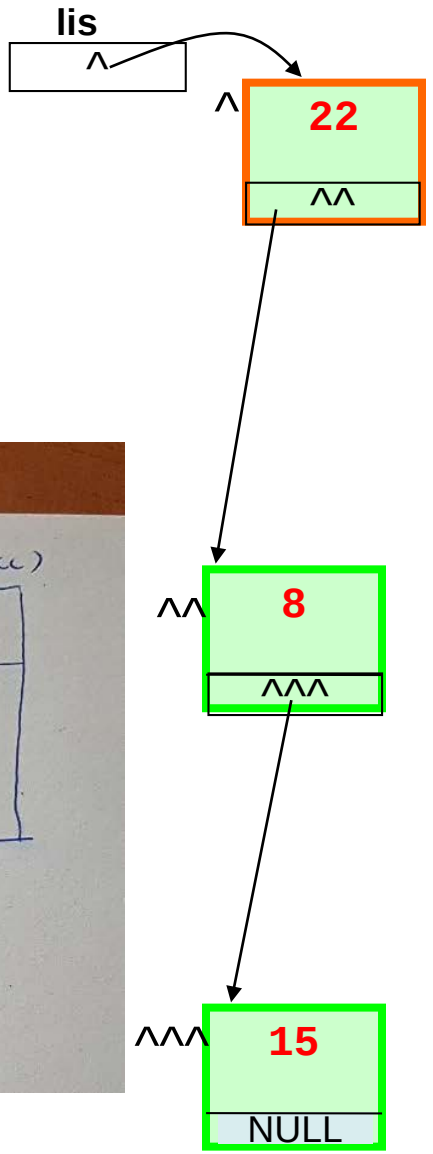


aggiungiUno(lis) RDA

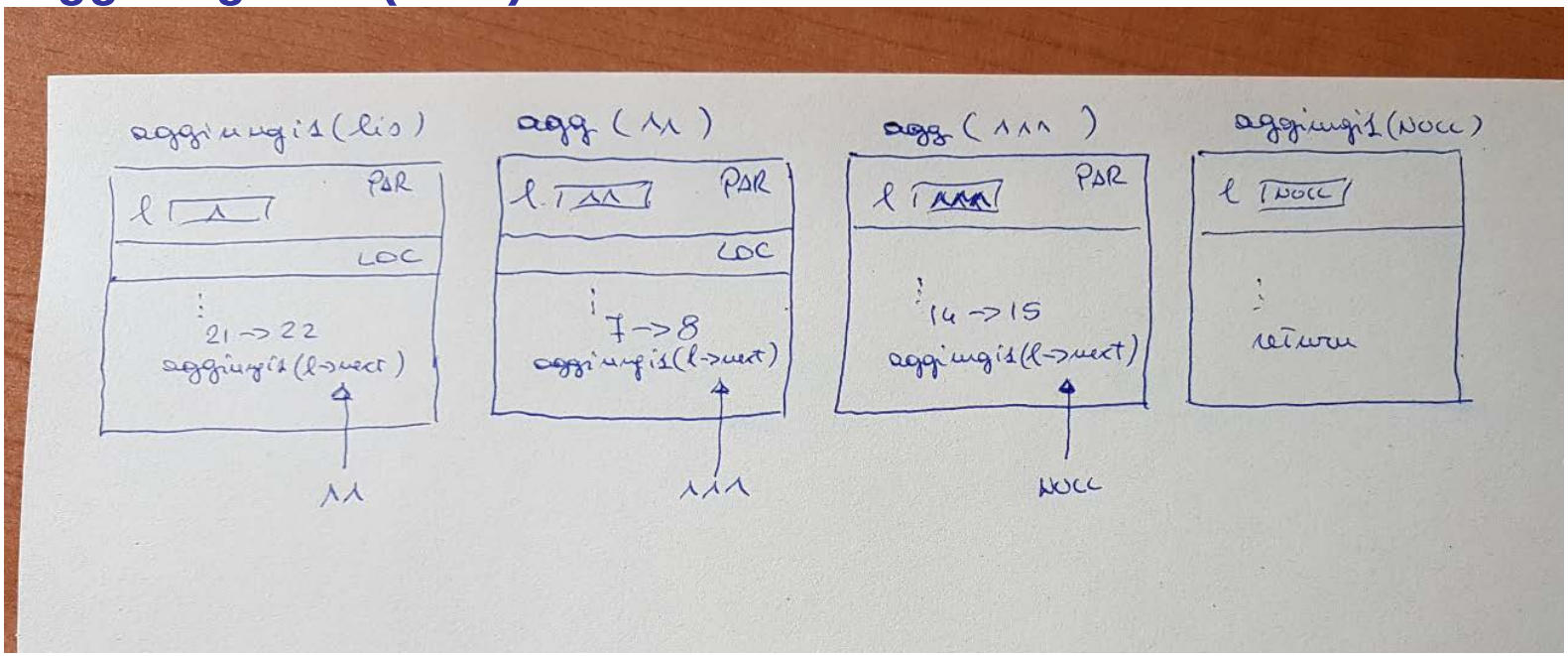


LISTE - *interpretazione induttiva*

```
void aggiungiUno (TipoLista l) {
  if (l) {
    l->info += 1;
    aggiungiUno(l->next);
  }
return;
}
```

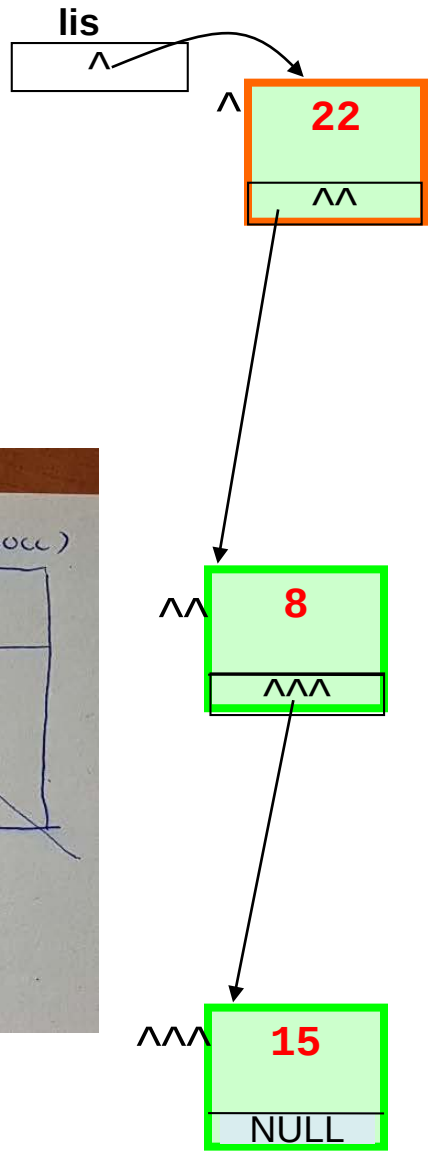


aggiungiUno(lis) RDA

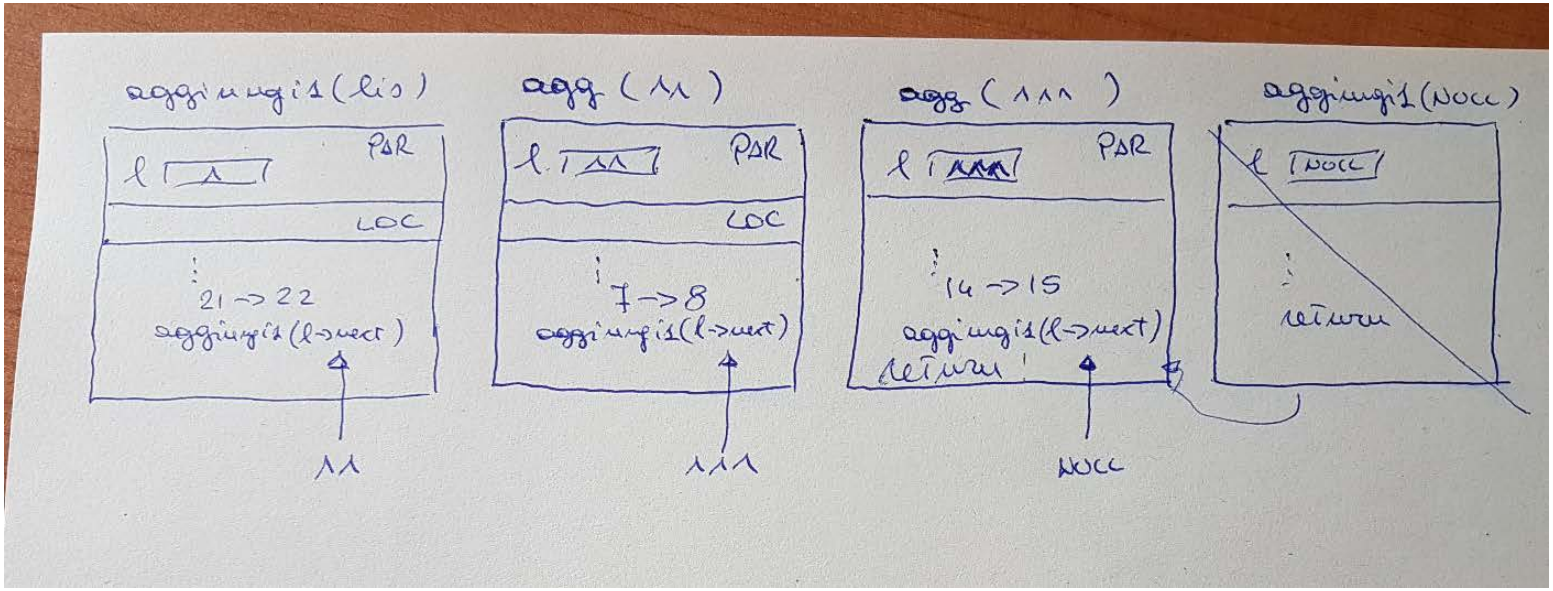


LISTE - *interpretazione induttiva*

```
void aggiungiUno (TipoLista l) {
  if (l) {
    l->info += 1;
    aggiungiUno(l->next);
  }
return;
}
```

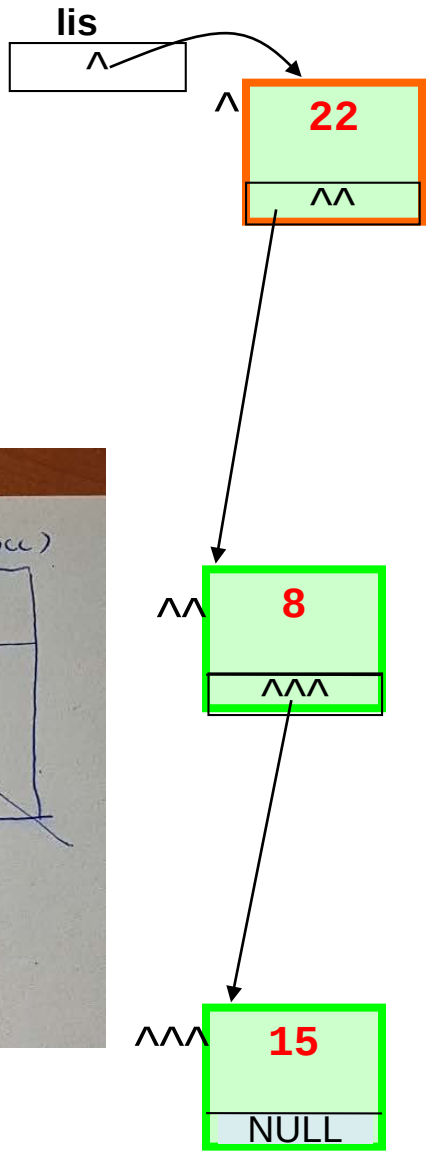


aggiungiUno(lis) RDA

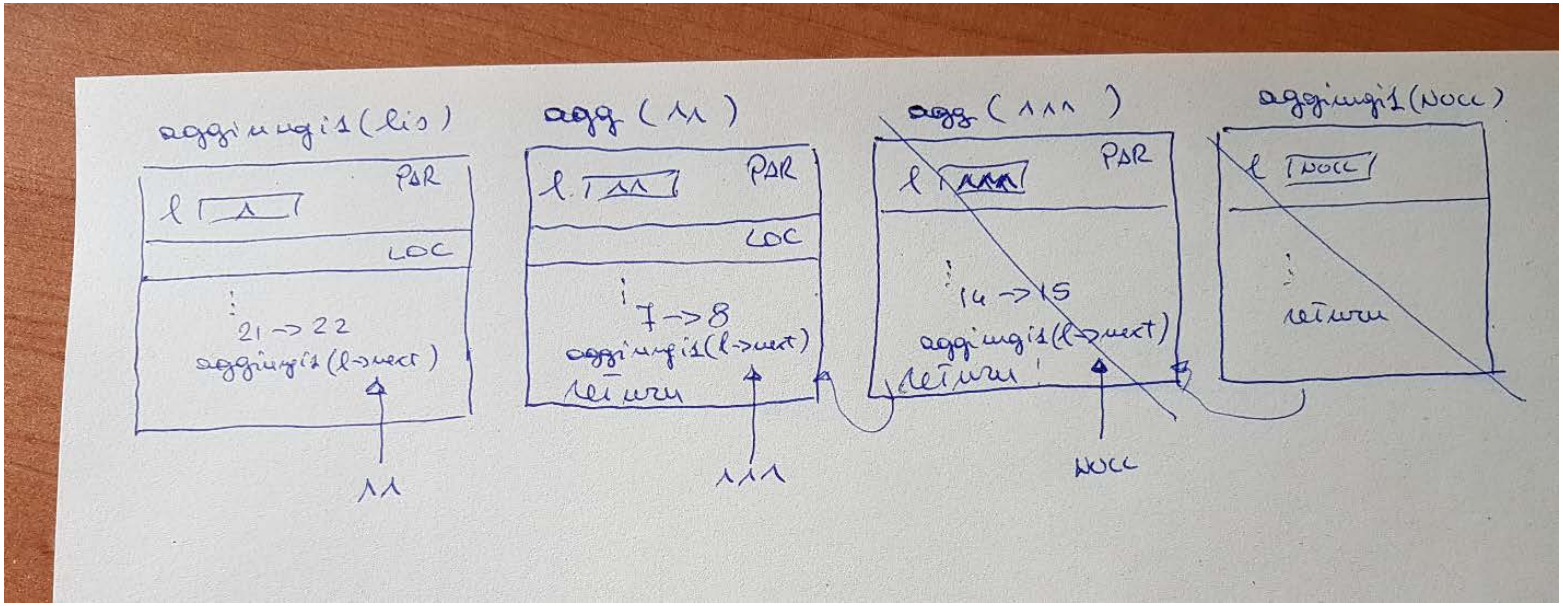


LISTE - *interpretazione induttiva*

```
void aggiungiUno (TipoLista l) {
  if (l) {
    l->info += 1;
    aggiungiUno(l->next);
  }
return;
}
```

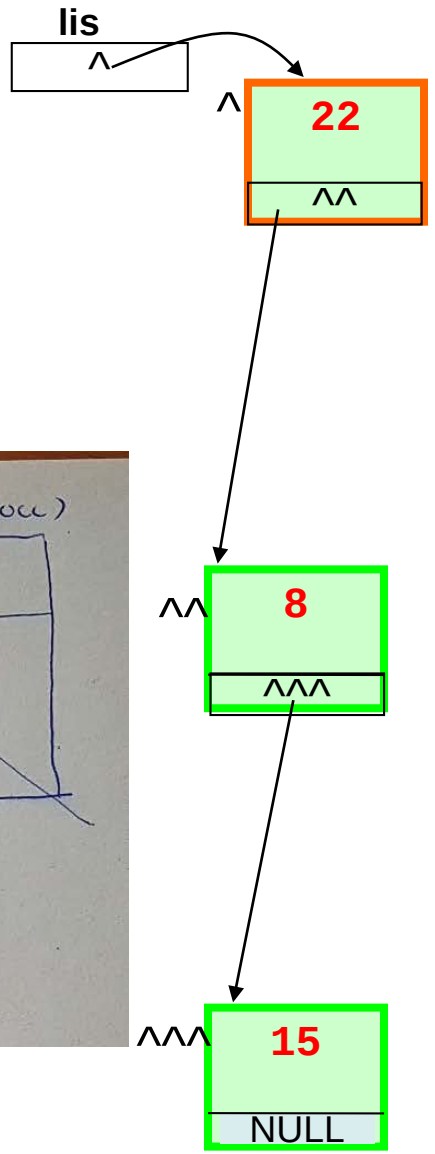


aggiungiUno(lis) RDA

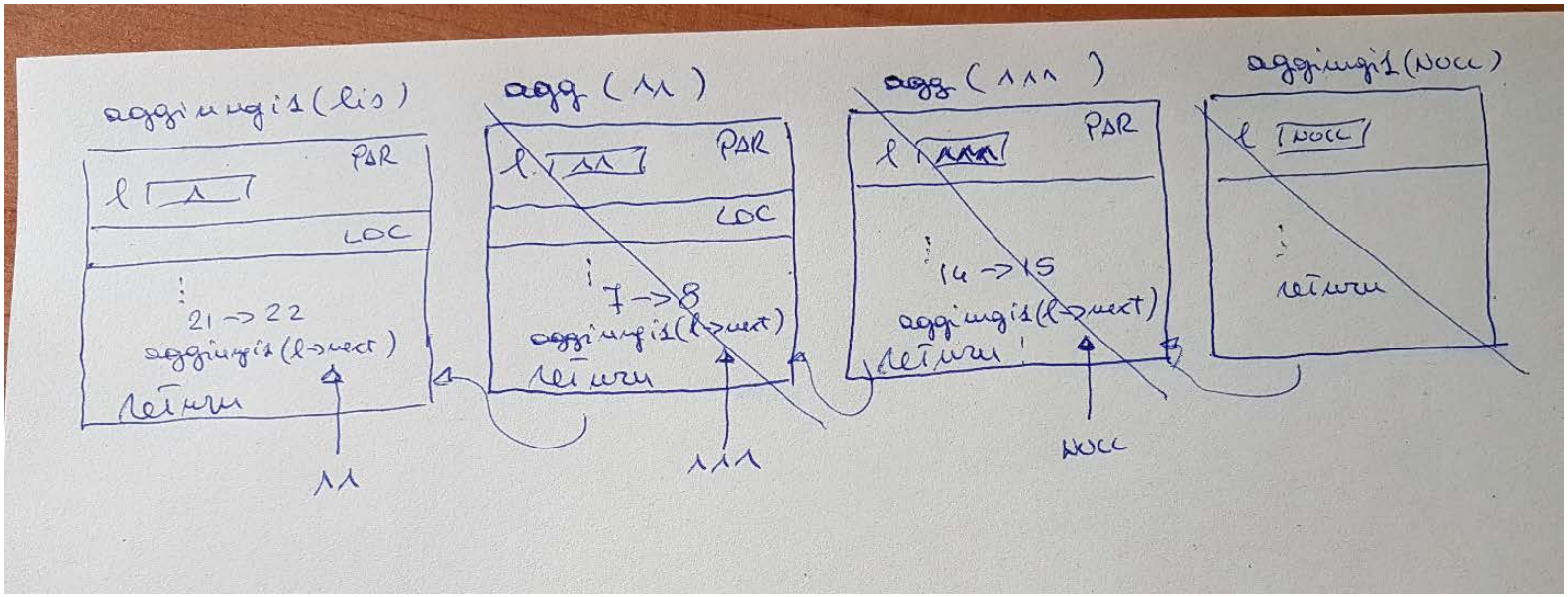


LISTE - *interpretazione induttiva*

```
void aggiungiUno (TipoLista l) {
  if (l) {
    l->info += 1;
    aggiungiUno(l->next);
  }
return;
}
```

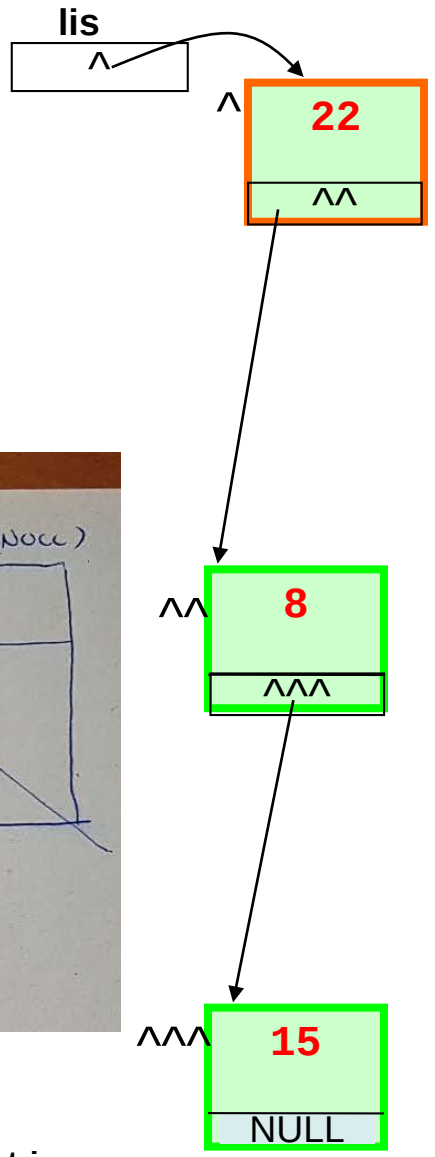


aggiungiUno(lis) RDA

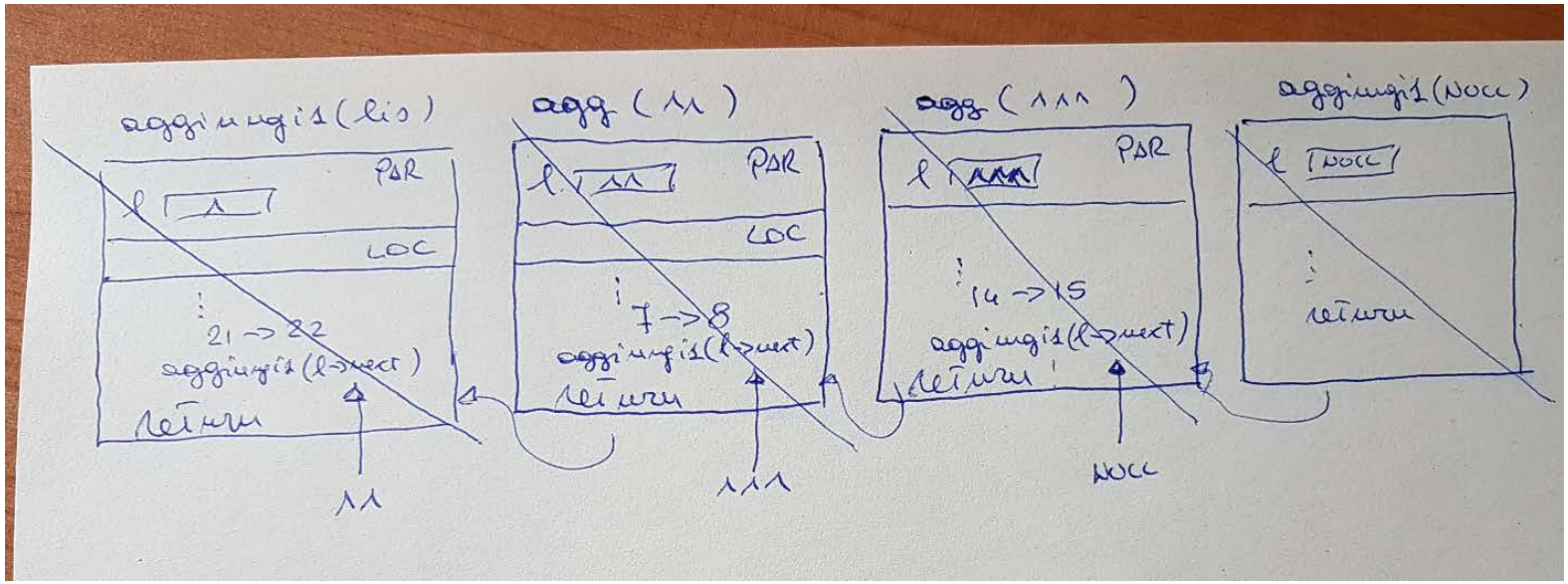


LISTE - *interpretazione induttiva*

```
void aggiungiUno (TipoLista l) {
  if (l) {
    l->info += 1;
    aggiungiUno(l->next);
  }
return;
}
```



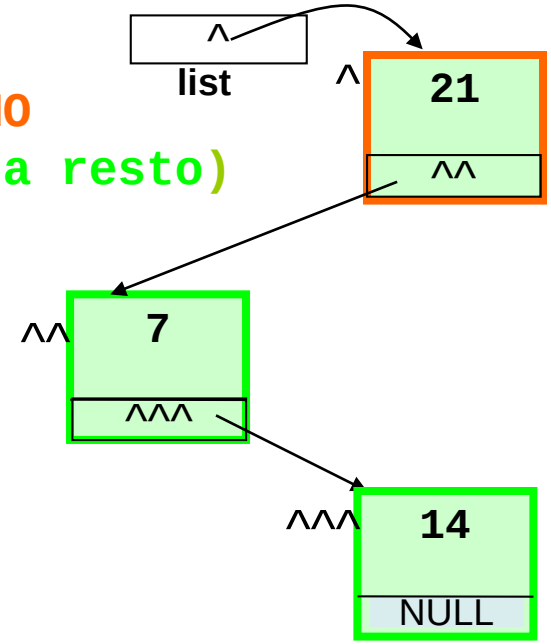
aggiungiUno(lis) RDA



Abbiamo avuto fino a quattro RDA allocati contemporaneamente (poi deallocati uno dopo l'altro dall'ultimo al primo che era stato allocato)

Stampa ricorsiva di una lista

```
STAMPA(LISTA) = { se LISTA NON VUOTA   1) output PRIMO
                                       2) STAMPA(lista resto)
                 senno`                NULLA
```

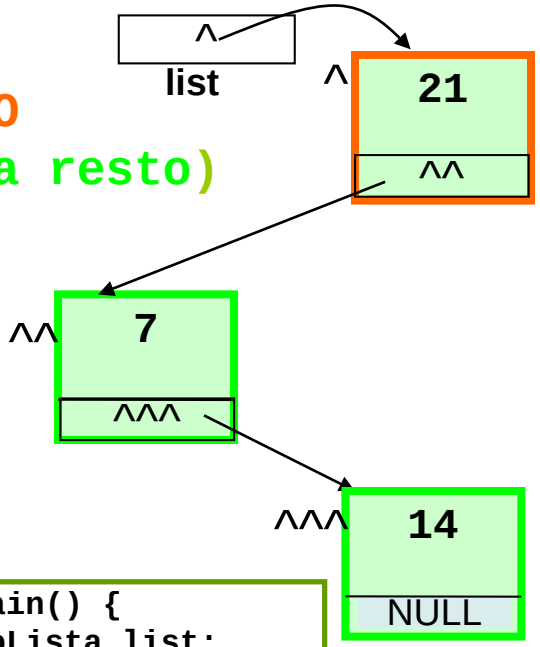



```
void stampaListaRic (TipoLista l) {
  if (l) {

                ☺

  }
return;
}
```

Stampa ricorsiva di una lista

STAMPA(LISTA) =

- se LISTA NON VUOTA → 1) output PRIMO; 2) STAMPA(lista resto)
- senno` → NULLA




```
void stampaListaRic (TipoLista l) {
  if (l) {
/* stampaElem(l->info); */
    printf("...%d...", l->info);
    stampaListaRic(l->next);
  }
return;
}
```

```
int main() {
  TipoLista list;
...
stampaListaRic(list);
...
return 0;
}
```

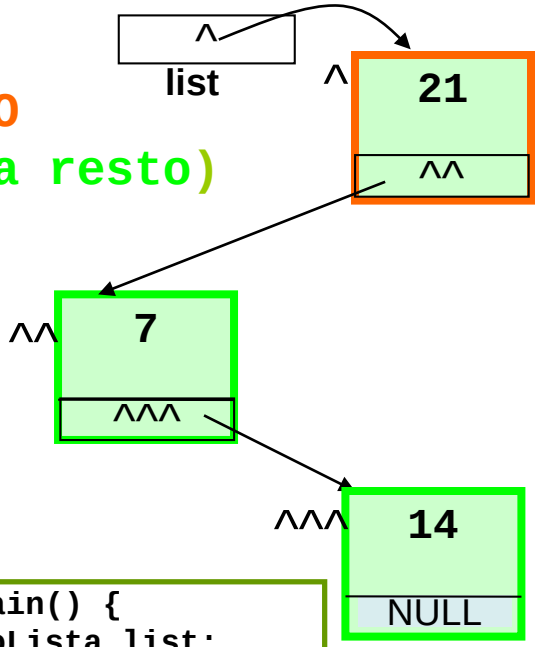
PRIMO

☺ chi sono questi due?

lista resto

Stampa ricorsiva di una lista

```
STAMPA(LISTA) = { se LISTA NON VUOTA   1) output PRIMO
                                       2) STAMPA(lista resto)
                { senno`               NULLA
```




```
void stampaListaRic (TipoLista l) {
  if (l) {
/* stampaElem(l->info); */
    printf("...%d...", l->info);
    stampaListaRic(l->next);
  }
return;
}
```

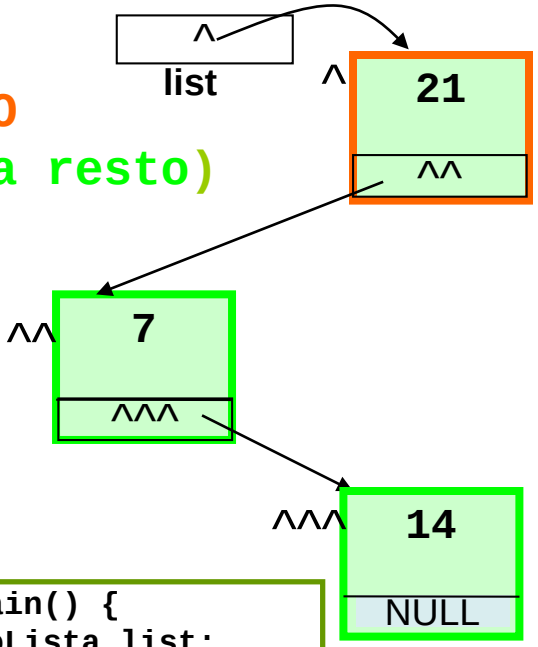
```
int main() {
  TipoLista list;
...
stampaListaRic(list);
...
return 0;
}
```

PRIMO

lista resto

Stampa ricorsiva di una lista

```
STAMPA(LISTA) = { se LISTA NON VUOTA   1) output PRIMO
                                       2) STAMPA(lista resto)
                 senno`                NULLA
```




```
void stampaListaRic (TipoLista l) {
  if (l) {
/* stampaElem(l->info); */
    printf("...%d...", l->info);
    stampaListaRic(l->next);
  }
return;
}
```

```
int main() {
  TipoLista list;
...
stampaListaRic(list);
...
return 0;
}
```

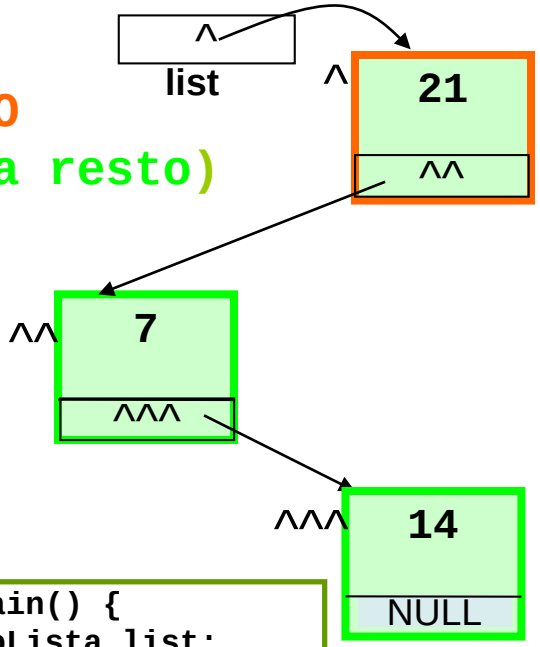
`stampaListaRic(list)`

```
l ^

printf("...%
stampa(l->next);
```

OUTPUT

Stampa ricorsiva di una lista

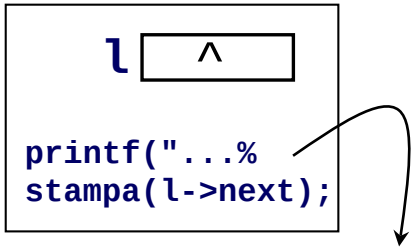
```
STAMPA(LISTA) = { se LISTA NON VUOTA   1) output PRIMO
                                       2) STAMPA(lista resto)
                { senno`               NULLA
```




```
void stampaListaRic (TipoLista l) {
  if (l) {
/* stampaElem(l->info); */
    printf("...%d...", l->info);
    stampaListaRic(l->next);
  }
return;
}
```

```
int main() {
  TipoLista list;
...
stampaListaRic(list);
...
return 0;
}
```

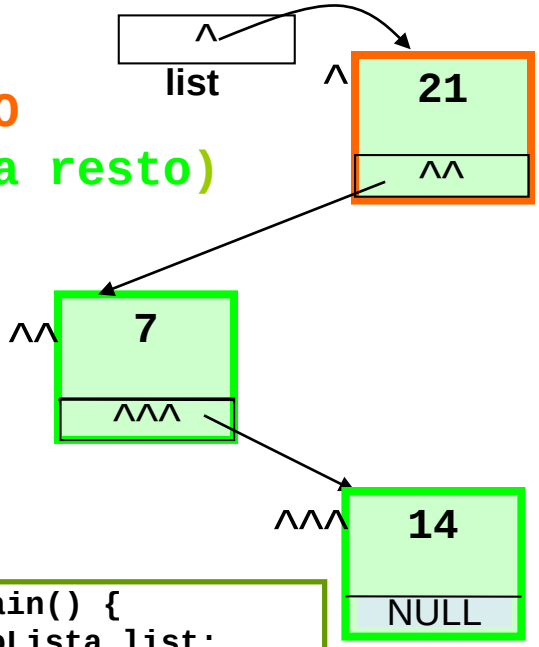
`stampa(list)` `stampa( ^^ )`



OUTPUT 21

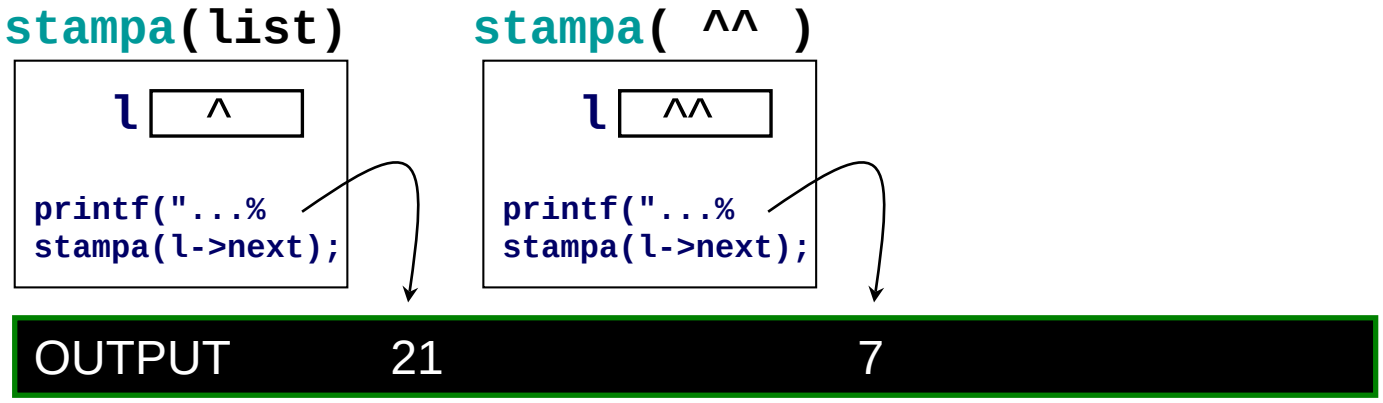
Stampa ricorsiva di una lista

```
STAMPA(LISTA) = { se LISTA NON VUOTA   1) output PRIMO
                                       2) STAMPA(lista resto)
                { senno`               NULLA
```




```
void stampaListaRic (TipoLista l) {
  if (l) {
/* stampaElem(l->info); */
    printf("...%d...", l->info);
    stampaListaRic(l->next);
  }
return;
}
```

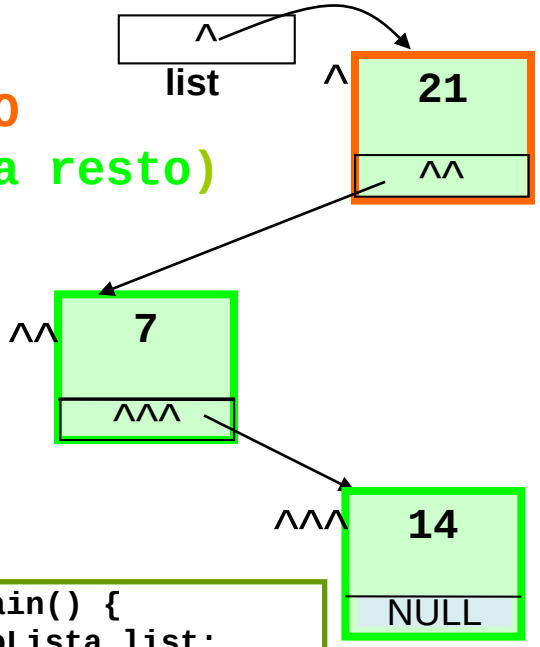
```
int main() {
  TipoLista list;
...
stampaListaRic(list);
...
return 0;
}
```



Stampa ricorsiva di una lista

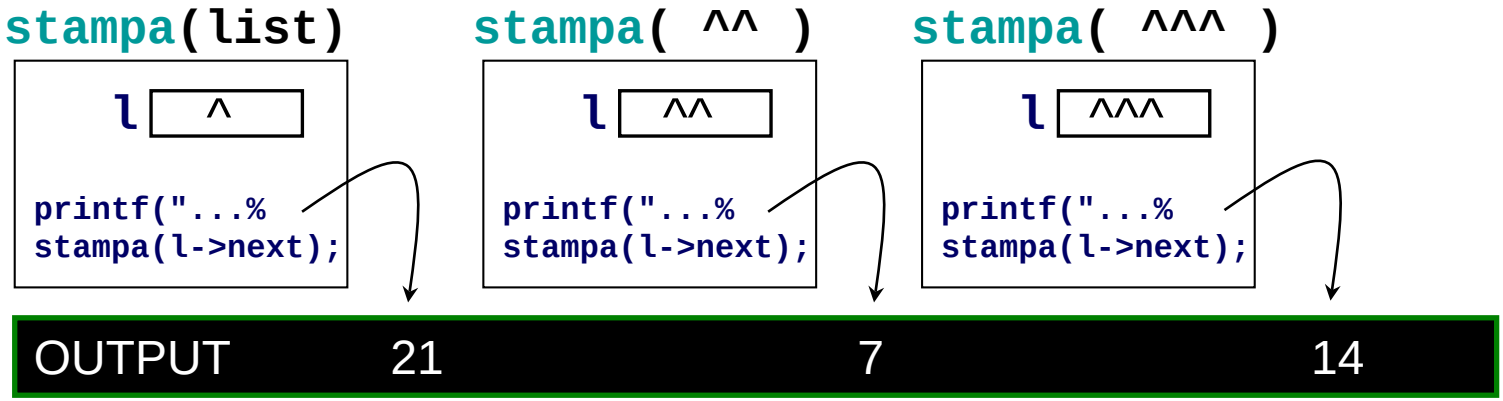
STAMPA(LISTA) =

- se LISTA NON VUOTA: 1) output PRIMO; 2) STAMPA(lista resto)
- senno`: NULLA



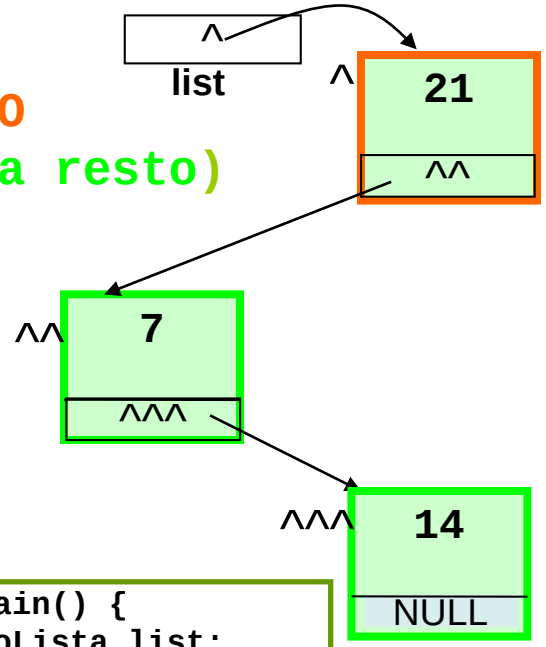

```
void stampaListaRic (TipoLista l) {
  if (l) {
/* stampaElem(l->info); */
    printf("...%d...", l->info);
    stampaListaRic(l->next);
  }
return;
}
```

```
int main() {
  TipoLista list;
...
stampaListaRic(list);
...
return 0;
}
```



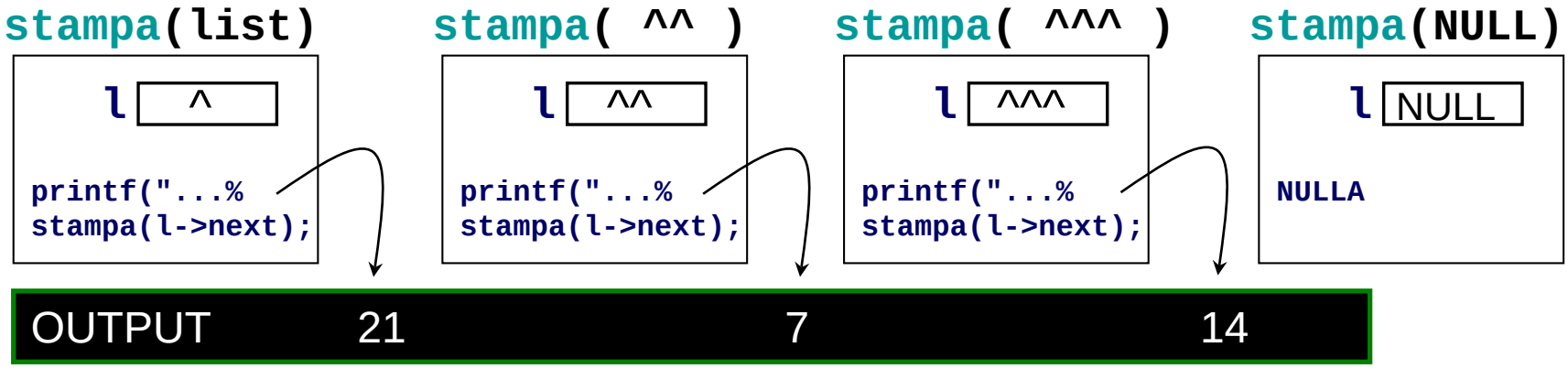
Stampa ricorsiva di una lista

```
STAMPA(LISTA) = { se LISTA NON VUOTA   1) output PRIMO
                                       2) STAMPA(lista resto)
                 senno`                NULLA
```




```
void stampaListaRic (TipoLista l) {
  if (l) {
/* stampaElem(l->info); */
    printf("...%d...", l->info);
    stampaListaRic(l->next);
  }
return;
}
```

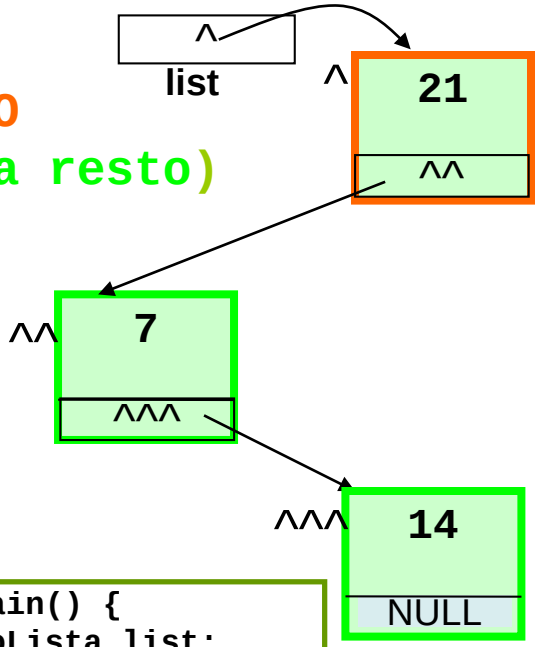
```
int main() {
  TipoLista list;
...
stampaListaRic(list);
...
return 0;
}
```



Stampa ricorsiva di una lista

STAMPA(LISTA) =

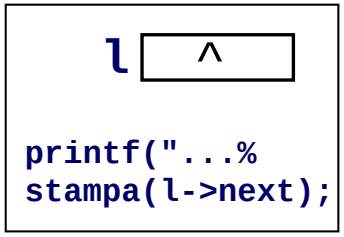
- se LISTA NON VUOTA: 1) output PRIMO; 2) STAMPA(lista resto)
- senno`: NULLA



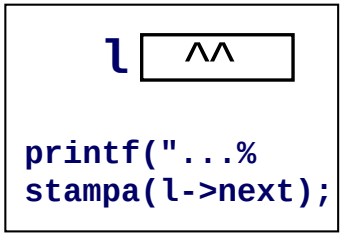

```
void stampaListaRic (TipoLista l) {
  if (l) {
/* stampaElem(l->info); */
    printf("...%d...", l->info);
    stampaListaRic(l->next);
  }
return;
}
```

```
int main() {
  TipoLista list;
...
stampaListaRic(list);
...
return 0;
}
```

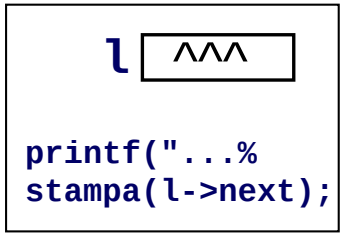
stampa(list)



stampa(^^)



stampa(^^^)

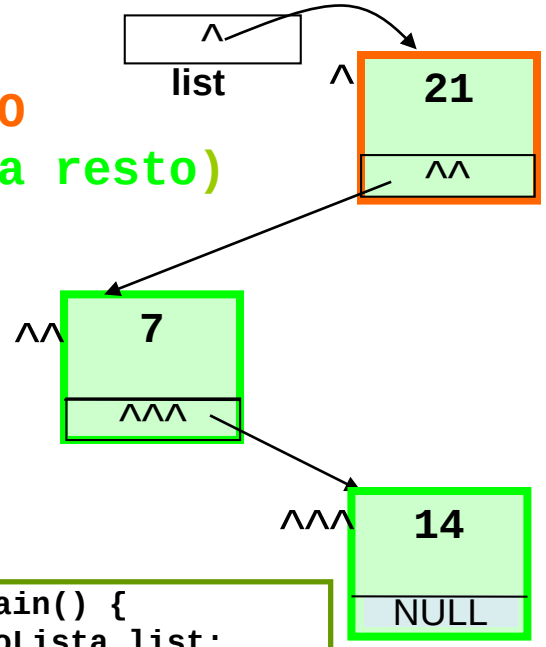


OUTPUT 21 7 14

Stampa ricorsiva di una lista

STAMPA(LISTA) =

- se LISTA NON VUOTA: 1) output PRIMO; 2) STAMPA(lista resto)
- senno`: NULLA




```
void stampaListaRic (TipoLista l) {
  if (l) {
/* stampaElem(l->info); */
    printf("...%d...", l->info);
    stampaListaRic(l->next);
  }
return;
}
```

```
int main() {
  TipoLista list;
...
stampaListaRic(list);
...
return 0;
}
```

stampa(list)

```
l ^
printf("...%
stampa(l->next);
```

stampa(^^)

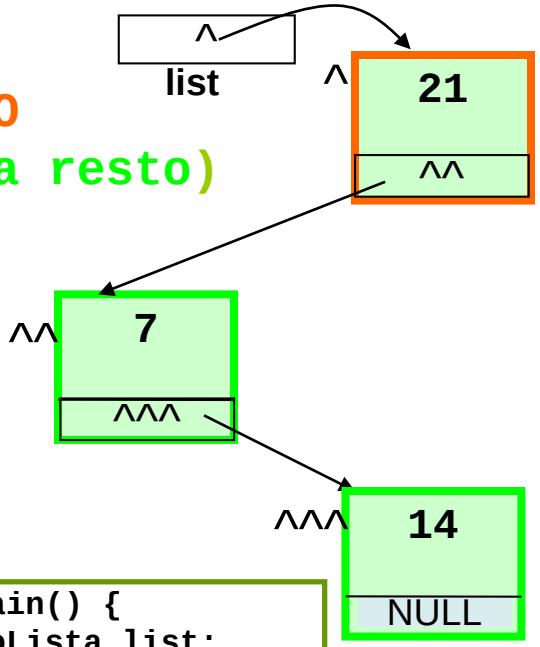
```
l ^^
printf("...%
stampa(l->next);
```

OUTPUT 21 7 14

Stampa ricorsiva di una lista

STAMPA(LISTA) =

- se LISTA NON VUOTA → 1) output PRIMO; 2) STAMPA(lista resto)
- senno` → NULLA




```
void stampaListaRic (TipoLista l) {
  if (l) {
/* stampaElem(l->info); */
    printf("...%d...", l->info);
    stampaListaRic(l->next);
  }
return;
}
```

```
int main() {
  TipoLista list;
...
stampaListaRic(list);
...
return 0;
}
```

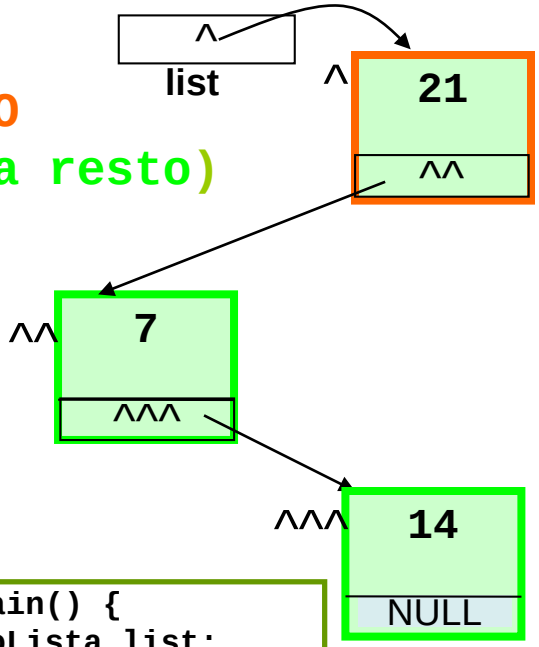
stampa(list)

```
l
printf("...%
stampa(l->next);
```

OUTPUT 21 7 14

Stampa ricorsiva di una lista

```
STAMPA(LISTA) = { se LISTA NON VUOTA   1) output PRIMO
                                       2) STAMPA(lista resto)
                { senno`               NULLA
```

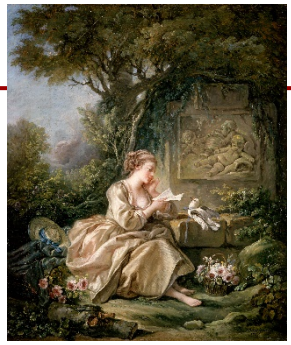



```
void stampaListaRic (TipoLista l) {
  if (l) {
/* stampaElem(l->info); */
    printf("...%d...", l->info);
    stampaListaRic(l->next);
  }
return;
}
```

```
int main() {
  TipoLista list;
...
stampaListaRic(list);
...
return 0;
}
```

OUTPUT 21 7 14

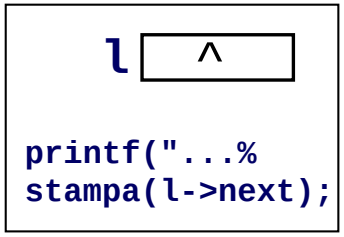
Stampa ricorsiva di una lista - con messaggio?



```
STAMPA(LISTA) = { se LISTA NON VUOTA   1) stampa PRIMO
                                       2) STAMPA(lista resto)
                 senno`  stampa "la lista e` VUOTA"?
```

```
void stampaListaRic (TipoLista l) {
  if (l) {     printf("...%d...", l->info);
               stampaListaRic(l->next);
  }
  else printf("Oh, la lista e` VUOTA");
return;
}
```


```
stampaListaRic(list)
```



☺ rda?

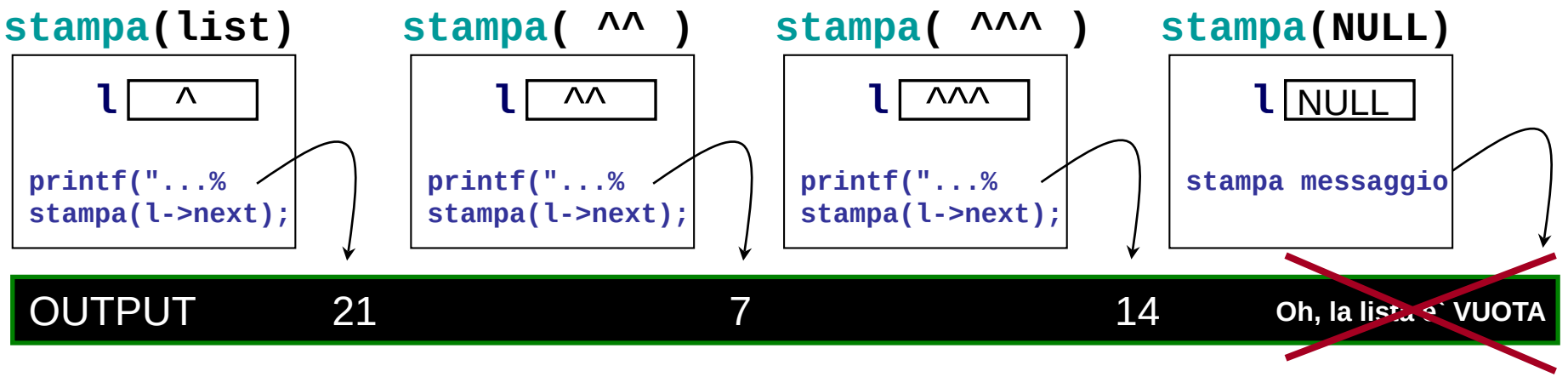
OUTPUT

Stampa ricorsiva di una lista - con messaggio?

```
STAMPA(LISTA) = { se LISTA NON VUOTA   1) stampa PRIMO
                                       2) STAMPA(lista resto)
                { senno`  stampa "la lista e` VUOTA"?
```

```
void stampaListaRic (TipoLista l) {
  if (l) {     printf("...%d...", l->info);
               stampaListaRic(l->next);
  }
  else printf("Oh, la lista e` VUOTA");
return;
}
```

meglio non mettere messaggi



Stampa ricorsiva di una lista - con messaggio?

```
) stampa PRIMO
) STAMPA(lista resto)
```

```
"la lista e` VUOTA"?
```

```
void s
  if (
  }
  else
return
}
```

```
{
l->info);
next);
```

meglio non mettere messaggi

```
stampa(
   l
printf("
stampa(l-
```

```
tampa( ^^^ )
   l  ^^^
rintf("...%
tampa(l->next);
```

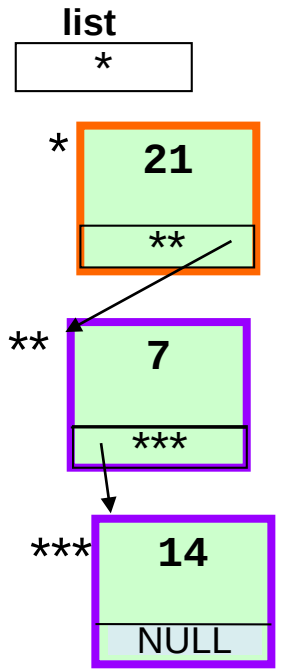
```
stampa(NULL)
sta
```

OUTPUT 21 7 14 ~~Lista VUOTA~~

Stampa ricorsiva di una lista

STAMPA(LISTA) = AL CONTRARIO

- se LISTA NON VUOTA: 1) STAMPA(lista resto) 2) stampa PRIMO
- senno`: NULLA




```
void stampaListaRic2 (TipoLista l) {
  if (l) {
     stampaListaRic(l->next);
     printf("...%d...", l->info);
  }
return;
}
```

```
void stampaListaRic (TipoLista l) {
  if (l) {
/* stampaElem(l->info); */
    printf("...%d...", l->info);
    stampaListaRic(l->next);
  }
return;
}
```

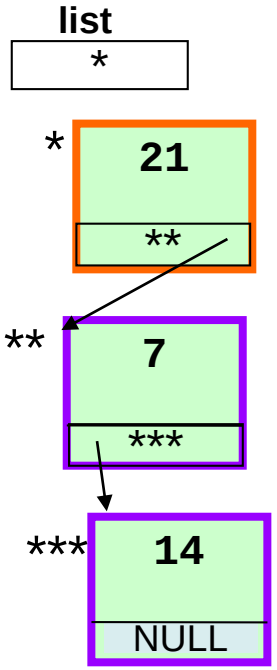
☺ rda?

OUTPUT

Stampa ricorsiva di una lista AL CONTRARIO

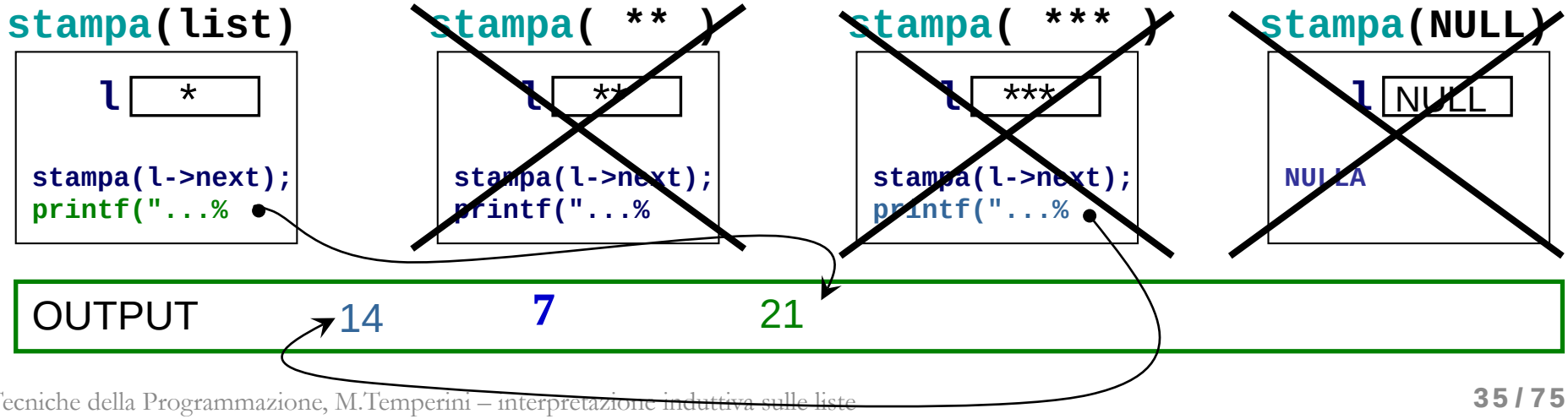
STAMPA(LISTA) = AL CONTRARIO

se LISTA NON VUOTA	1) STAMPA(lista resto) 2) stampa PRIMO AL CONTRARIO
senno`	NULLA



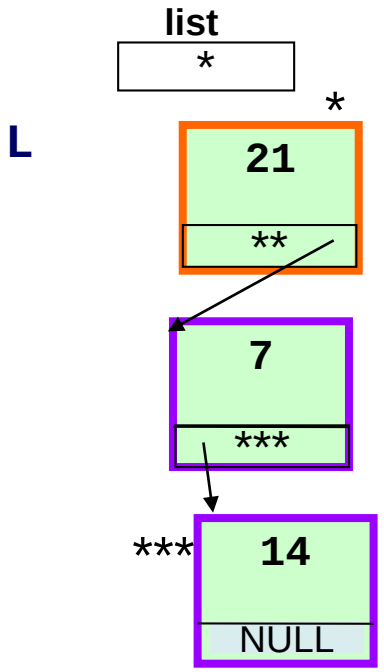

```
void stampaListaRic2 (TipoLista l) {
  if (l) {
     stampaListaRic(l->next);
     printf("...%d...", l->info);
  }
return;
}
```

```
void stampaListaRic (TipoLista l) {
  if (l) {
/* stampaElem(l->info); */
    printf("...%d...", l->info);
    stampaListaRic(l->next);
  }
return;
}
```



Calcolo del numero di nodi di una lista

una lista non vuota ha almeno un elemento (+ quelli della lista resto)

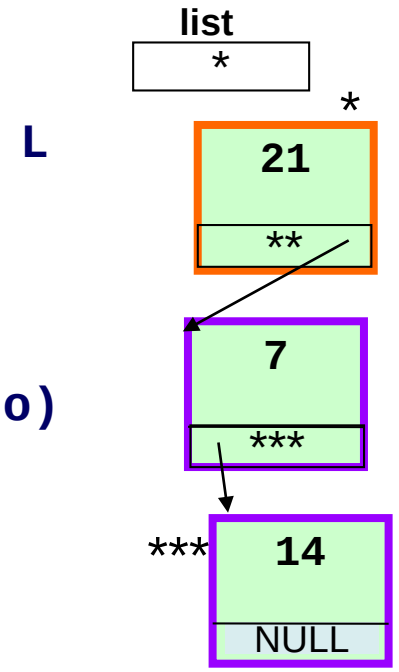


numNodi(L) =

- se L NON VUOTA — return what?
- senno` — return what?

Calcolo del numero di nodi di una lista

una lista non vuota ha almeno un elemento (+ quelli della lista resto)



$$\text{numNodi}(L) = \begin{cases} \text{se } L \text{ NON VUOTA} & \text{return } 1 + \text{numNodi(listaresto)} \\ \text{senno`} & \text{return } 0 \end{cases}$$

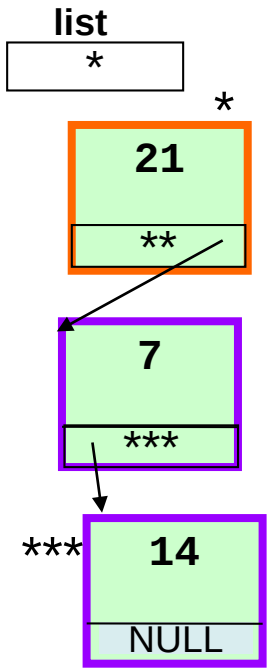
```
int numNodi (TipoLista l) {
  if (l)
    return (1 + numNodi(l->next));
  else return 0;
}
```

eseguire l'esecuzione simulata come per gli esempi precedenti (mostrando i RDA) !!

Calcolo del numero di nodi di una lista

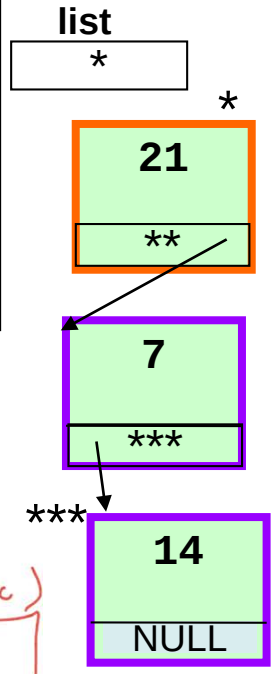
```
int numNodi (TipoLista l) {
  if (l)
    return (1 + numNodi(l->next));
  else return 0;
}
```

numNodi(list)……. RDA

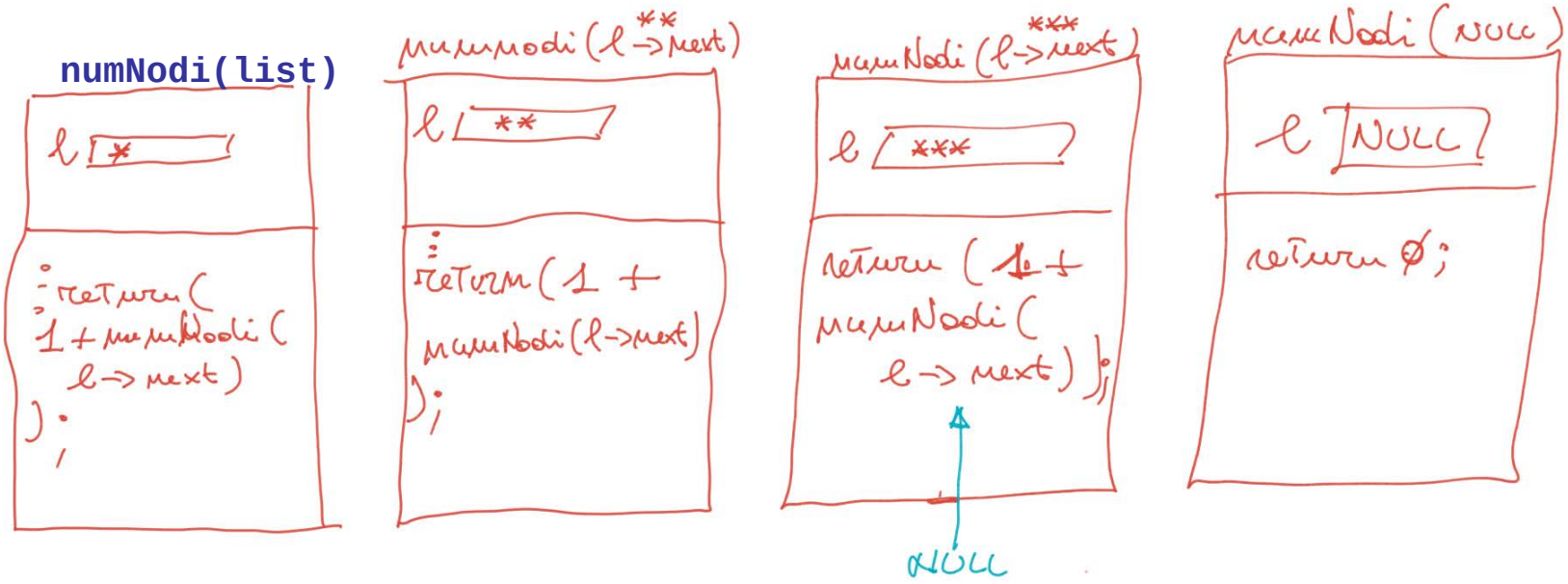


Calcolo del numero di nodi di una lista

```
int numNodi (TipoLista l) {
  if (l)
    return (1 + numNodi(l->next));
  else return 0;
}
```

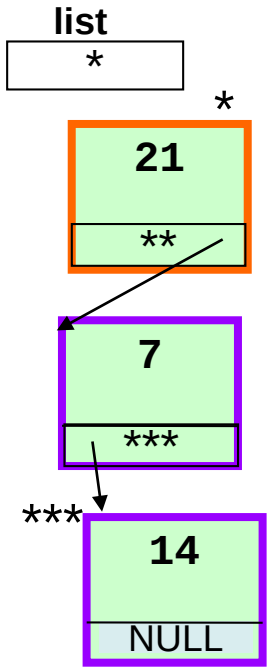


RDA

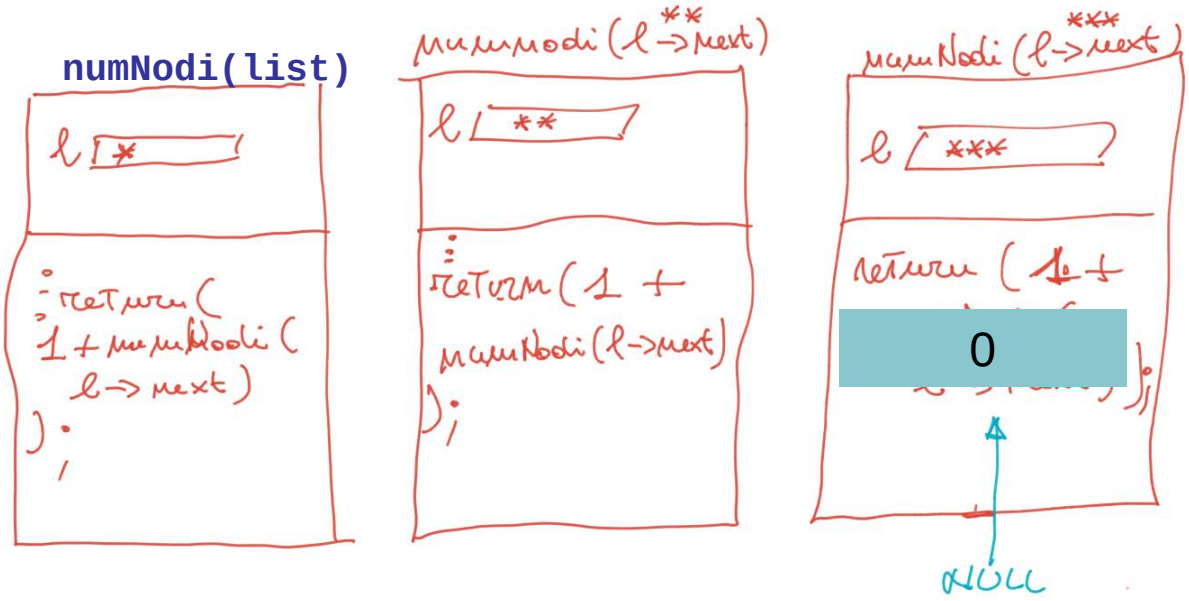


Calcolo del numero di nodi di una lista

```
int numNodi (TipoLista l) {
  if (l)
    return (1 + numNodi(l->next));
  else return 0;
}
```

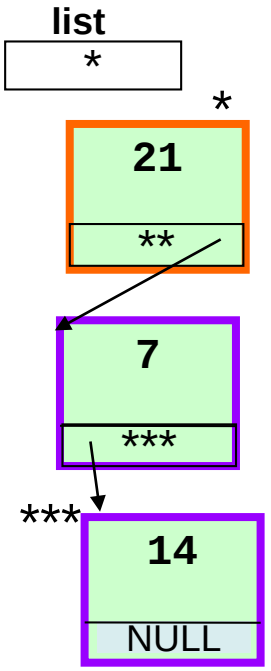


RDA

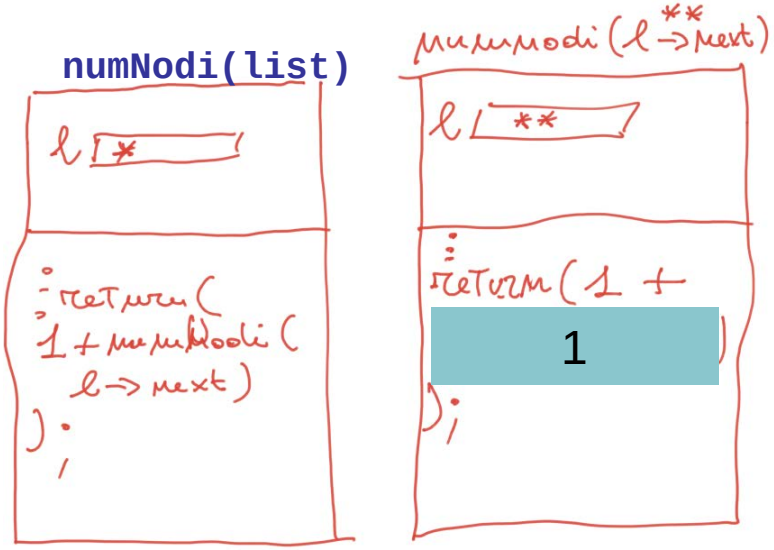


Calcolo del numero di nodi di una lista

```
int numNodi (TipoLista l) {
  if (l)
    return (1 + numNodi(l->next));
  else return 0;
}
```

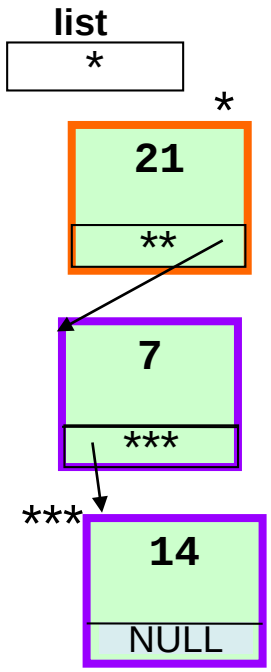


RDA

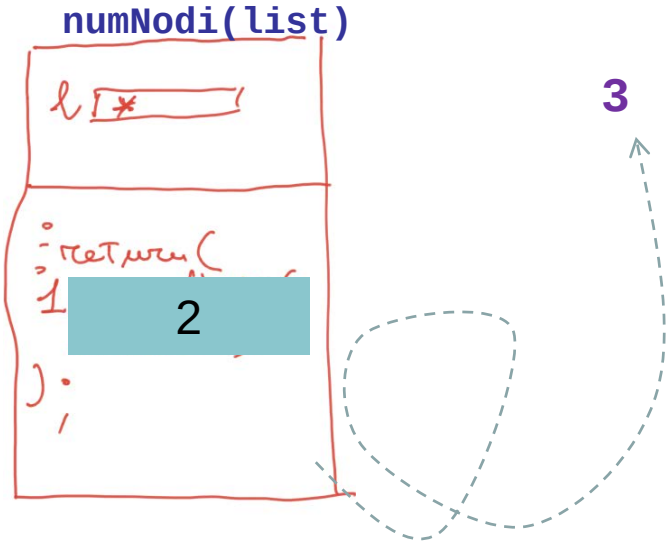


Calcolo del numero di nodi di una lista

```
int numNodi (TipoLista l) {
  if (l)
    return (1 + numNodi(l->next));
  else return 0;
}
```

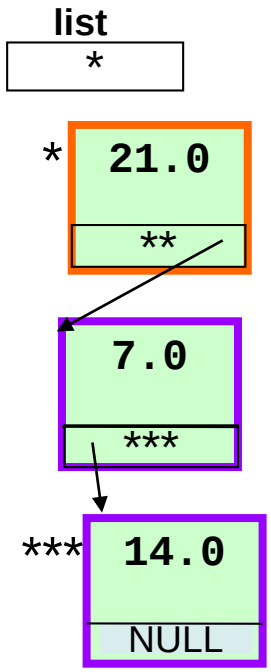


RDA



Calcolo ☺☺☺☺☺☺☺☺ di una lista

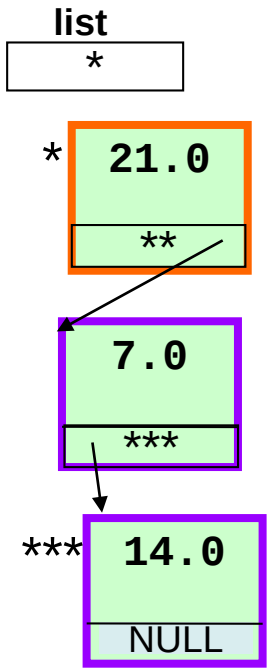
```
double ☺☺☺☺☺☺☺☺(TipoLista l) {
  if (l)
    return (l->info + ☺☺☺☺☺☺☺☺(l->next));
  else return 0.0;
}
```



scrivere l'algoritmo

dare un nome alla funzione

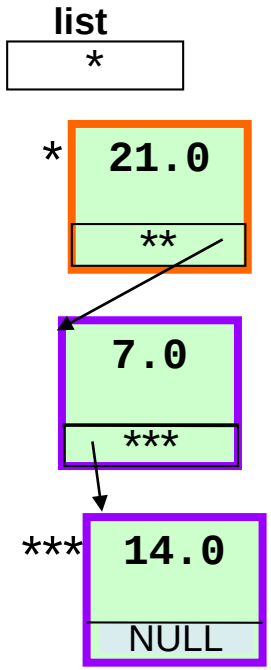
```
double ☺☺☺☺☺☺☺☺☺☺ (TipoLista l) {
  if (l)
    return (l->info + ☺☺☺☺☺☺☺☺☺☺ (l->next));
  else return 0.0;
}
```



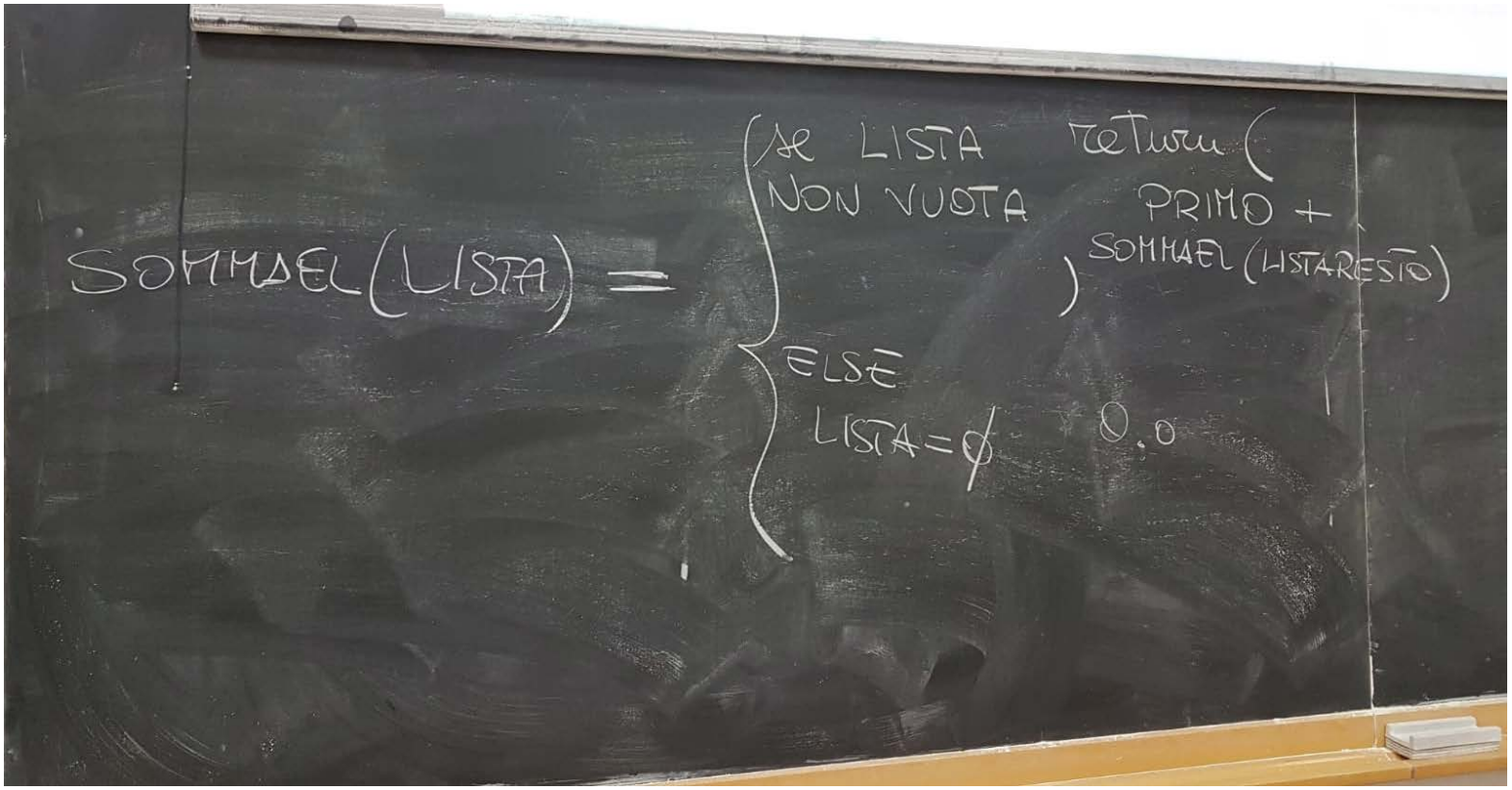
algoritmo?

☺(L) = { L VUOTA → 0.0 ; ≠ VUOTA → primo + ☺(restolista) }

```
double sommaElementi (TipoLista l) {
  if (l)
    return (l->info + sommaElementi(l->next));
  else return 0.0;
}
```



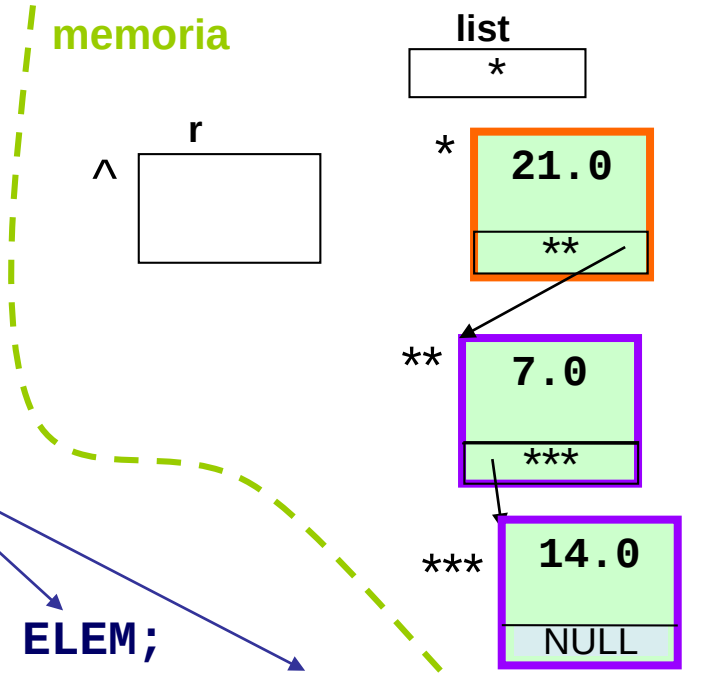
algoritmo?



Calcolo somma elementi (ALTERNATIVA) 1/2

Invece di fornire la somma come risultato, la si gestisce come parametro di output, *passato attraverso tutte le chiamate ricorsive*

```
int main() {
  TipoElem r;    TipoLista list;
... Azzera r …
sommaElementi (list, &r);
...
return 0;
}
```



side effect

r

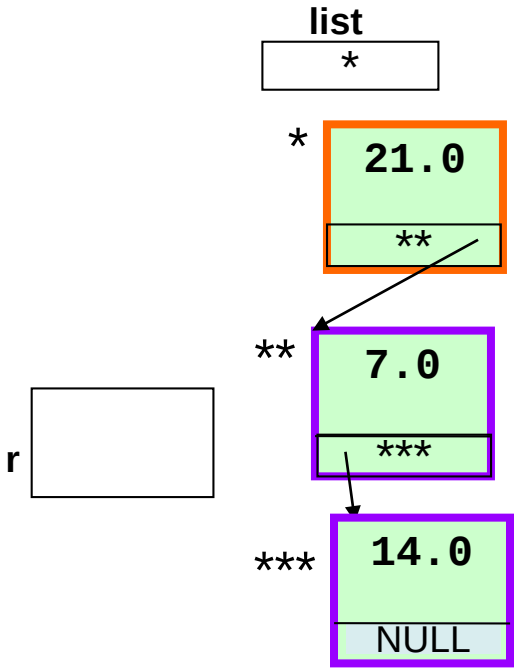
```
sommaElementi(LISTA, ELEM) =

    =  { se LISTA NON VUOTA  -  aggiungi primo a ELEM;
                             -  sommaElementi(LISTARESTO, ELEM)

       { senno`              return e basta
```

Calcolo somma elementi (ALTERNATIVA) 2/2

```
sommaElementi(LISTA, ELEM) =

    =  { se LISTA NON VUOTA  -  aggiungi primo a ELEM;
                             -  sommaElementi(LISTARESTO, ELEM)
       { senno`
```

```
void sommaElementi (                    ,
                                         ){

  if (l) {


  }
return;
}
```

es. somma su lista di punti



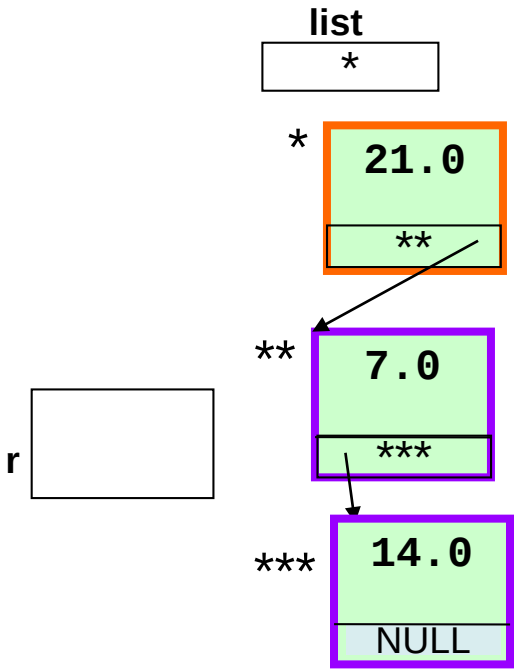

```
sommaElementi(LISTARESTO, ELEM)
sommaElementi(list, &r)
```

Calcolo somma elementi (ALTERNATIVA) 2/2

```
sommaElementi(LISTA, ELEM) =

      =  { se LISTA NON VUOTA  -  aggiungi primo a ELEM;
                               -  sommaElementi(LISTARESTO, ELEM)
         { senno`
```

```
void sommaElementi (TipoLista l,
                                    ){
  if (l) {


  }
return;
}
```

es. somma su lista di punti



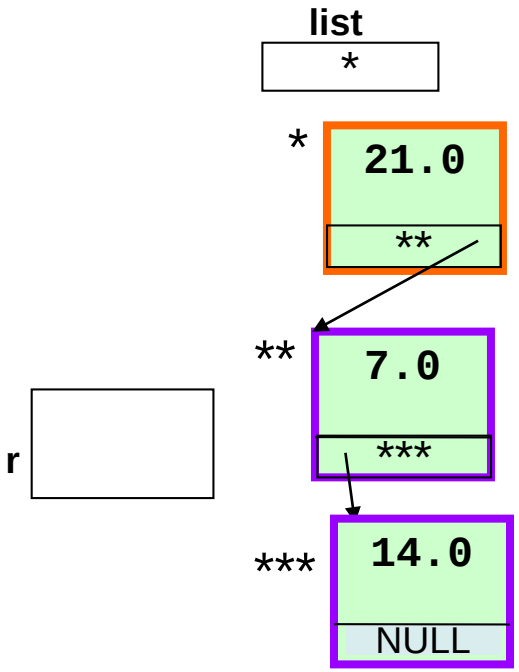

```
sommaElementi(list, &r)
```

Calcolo somma elementi (ALTERNATIVA) 2/2

```
sommaElementi(LISTA, ELEM) =

      =  { se LISTA NON VUOTA  -  aggiungi primo a ELEM;
                               -  sommaElementi(LISTARESTO, ELEM)
         { senno`
```

```
void sommaElementi (TipoLista l,
                    TipoElem * pel){
  if (l) {


  }
return;
}
```

es. somma su lista di punti



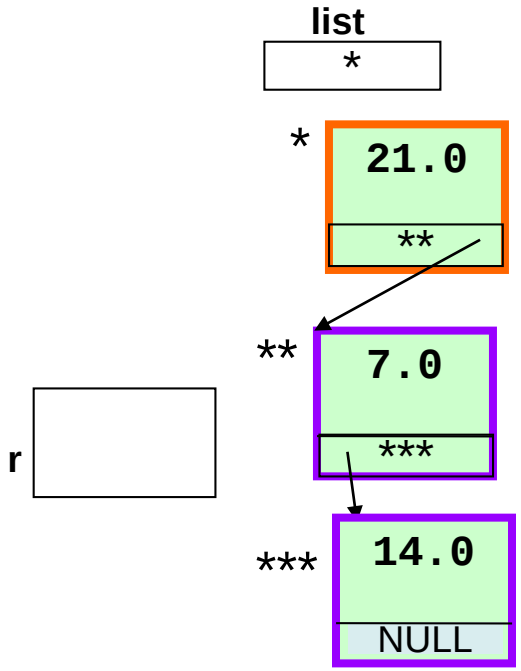

```
sommaElementi(list, &r)
```

Calcolo somma elementi (ALTERNATIVA) 2/2

```
sommaElementi(LISTA, ELEM) =

      =  { se LISTA NON VUOTA  -  aggiungi primo a ELEM;
         { senno`              -  sommaElementi(LISTARESTO, ELEM)
```

```
void sommaElementi (TipoLista l,
                    TipoElem * pel){
  if (l) {
    aggiungi (l->info, pel);

  }
return;
}
```

es. somma su lista di punti



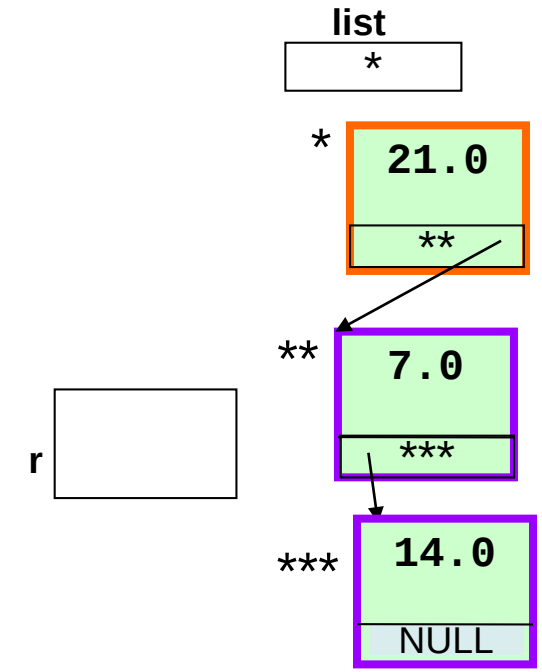

```
sommaElementi(list, &r)
```

Calcolo somma elementi (ALTERNATIVA) 2/2

```
sommaElementi(LISTA, ELEM) =

      =  { se LISTA NON VUOTA  -  aggiungi primo a ELEM;
                               -  sommaElementi(LISTARESTO, ELEM)
         { senno`
```

```
void sommaElementi (TipoLista l,
                    TipoElem * pel){
  if (l) {
    aggiungi (l->info, pel);
    sommaElementi(l->next, pel)
  }
return;
}
```

es. somma su lista di punti



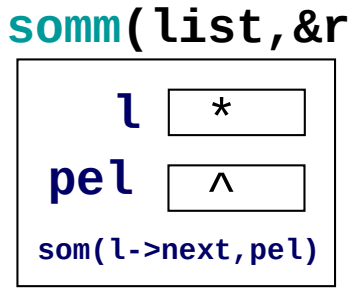

```
sommaElementi(list, &r)
```

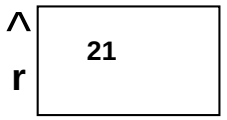
Calcolo somma elementi (ALTERNATIVA) 2/2

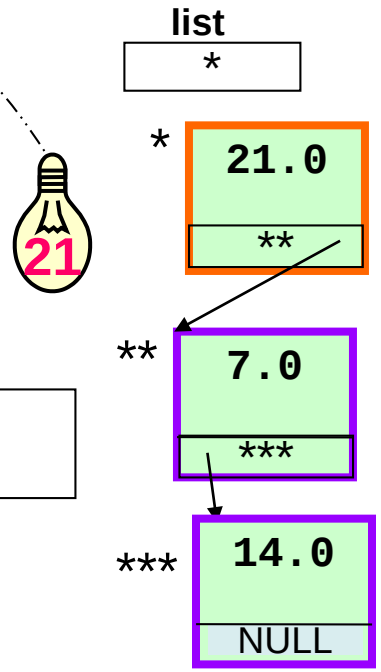
```
sommaElementi(LISTA, ELEM) =

    =  { se LISTA NON VUOTA  -  aggiungi primo a ELEM;
                             -  sommaElementi(LISTARESTO, ELEM)
       { senno`              return e basta
```

```
void sommaElementi (TipoLista l,
                    TipoElem * pel){
  if (l) {
    aggiungi (l->info, pel);
    sommaElementi(l->next, pel)
  }
return;
}
```

NB usiamo void aggiungi(TipoElem, TipoElem *)







es. somma su lista di punti

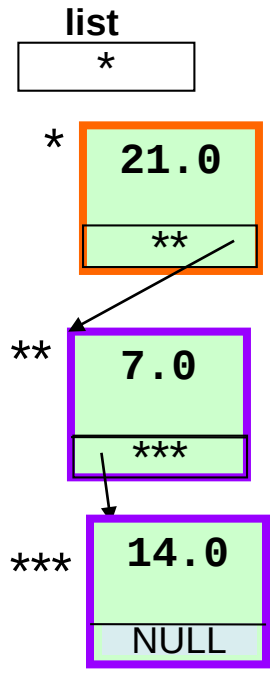
Calcolo somma elementi (ALTERNATIVA) 2/2

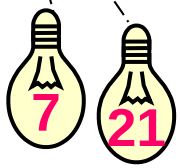
```
sommaElementi(LISTA, ELEM) =

      =  { se LISTA NON VUOTA  -  aggiungi primo a ELEM;
                               -  sommaElementi(LISTARESTO, ELEM)
         { senno`              return e basta
```

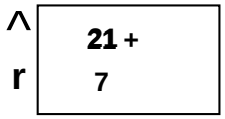
```
void sommaElementi (TipoLista l,
                    TipoElem * pel){
  if (l) {
    aggiungi (l->info, pel);
    sommaElementi(l->next, pel)
  }
return;
}
```

NB usiamo void aggiungi(TipoElem, TipoElem *)

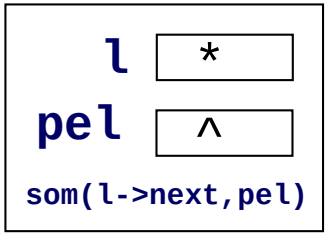
es. somma su lista di punti



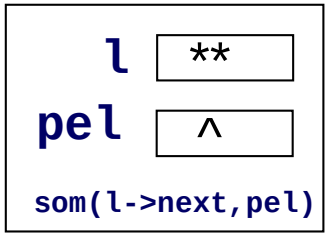




somm(list,&r)



somm(**,pel)



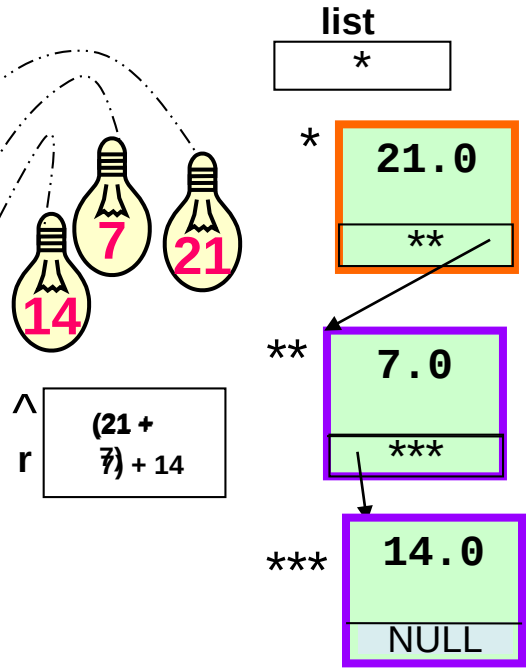
Calcolo somma elementi (ALTERNATIVA) 2/2

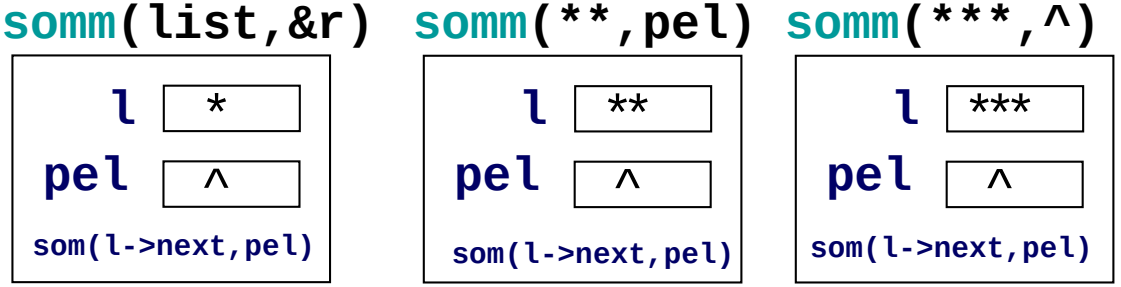
```
sommaElementi(LISTA, ELEM) =

    =  { se LISTA NON VUOTA  -  aggiungi primo a ELEM;
                             -  sommaElementi(LISTARESTO, ELEM)
       { senno`              return e basta
```

```
void sommaElementi (TipoLista l,
                    TipoElem * pel){
  if (l) {
    aggiungi (l->info, pel);
    sommaElementi(l->next, pel)
  }
return;
}
```

NB usiamo void aggiungi(TipoElem, TipoElem *)

es. somma su lista di punti





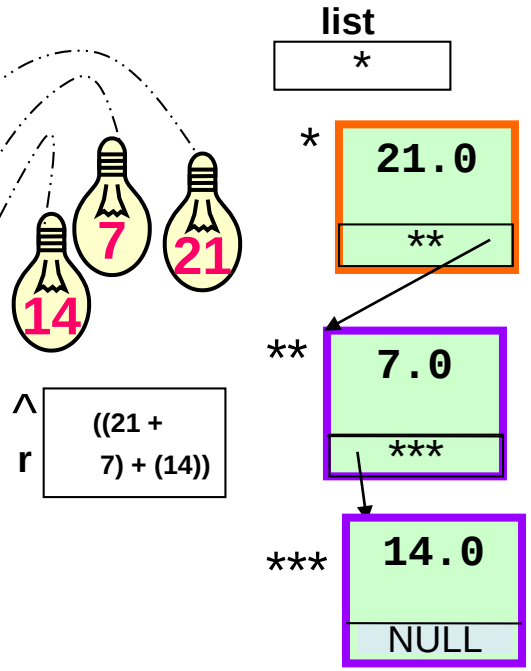
Calcolo somma elementi (ALTERNATIVA) 2/2

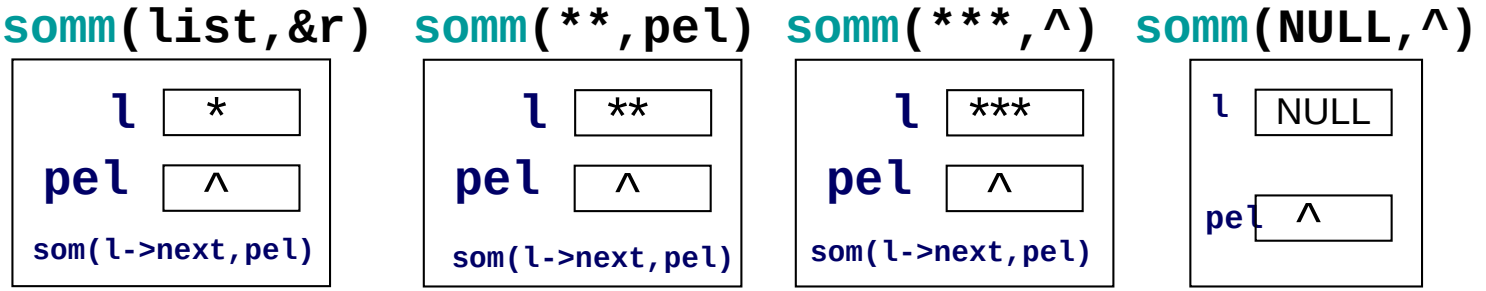
```
sommaElementi(LISTA, ELEM) =

    =  { se LISTA NON VUOTA  -  aggiungi primo a ELEM;
                             -  sommaElementi(LISTARESTO, ELEM)
       { senno`              return e basta
```

```
void sommaElementi (TipoLista l,
                    TipoElem * pel){
  if (l) {
    aggiungi (l->info, pel);
    sommaElementi(l->next, pel)
  }
return;
}
```

NB usiamo void aggiungi(TipoElem, TipoElem *)

es. somma su lista di punti



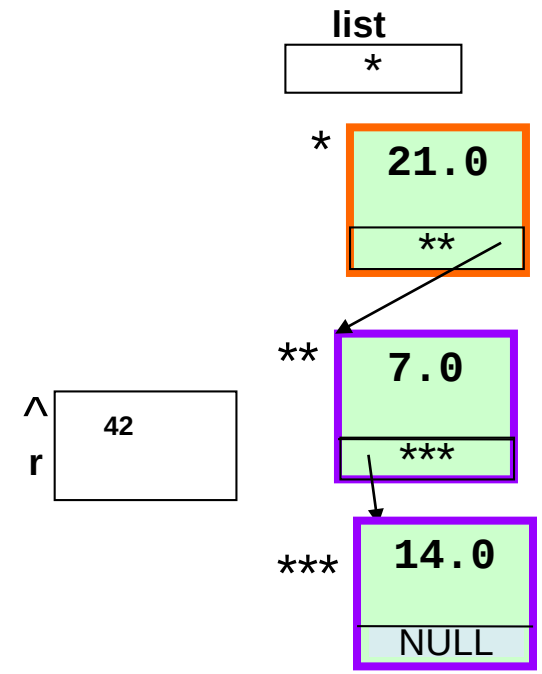


Calcolo somma elementi (ALTERNATIVA) 2/2

```
sommaElementi(LISTA, ELEM) =

      = { se LISTA NON VUOTA  -  aggiungi primo a ELEM;
                              -  sommaElementi(LISTARESTO, ELEM)
        { senno`              return e basta
```

```
void sommaElementi (TipoLista l,
                    TipoElem * pel){
  if (l) {
    aggiungi (l->info, pel);
    sommaElementi(l->next, pel)
  }
return;
}
```

es. somma su lista di punti



Calcolo media elementi in lista (1/3)

(TipoElem = int)

funzione non ricorsiva: restituisce direttamente la media, data la lista (per evitare che l'utilizzatore della funzione debba eseguire chiamate "innaturalmente complicate")

le due variabili locali di mediaLista() vengono passate (tramite INDIRIZZO) alla funzione `calcola()` che eseguira` su di loro dei side effect, in modo che al ritorno da questa chiamata di `calcola()` le due variabili locali di mediaLista() contengano i valori della somma complessiva dei nodi della lista e del numero di nodi in lista, necessari per calcolare la media

```
float mediaLista (TipoLista l) {

   int somma=0, numeroNodi=0;

  calcola (l, &somma, &numeroNodi);



}
```

Calcolo media elementi in lista (1/3)

(TipoElem = int)

funzione non ricorsiva: restituisce direttamente la media, data la lista (per evitare che l'utilizzatore della funzione debba eseguire chiamate "innaturalmente complicate")

```
float mediaLista (TipoLista l) {

   int somma=0, numeroNodi=0;

  calcola (l, &somma, &numeroNodi);
  if (numeroNodi)
     return (float)somma / numeroNodi;
  else return 0.0;
}
```

Calcolo media elementi in lista (1/3)

(TipoElem = int)

funzione non ricorsiva: restituisce direttamente la media, data la lista (per evitare che l'utilizzatore della funzione debba eseguire chiamate "innaturalmente complicate")

le due variabili locali di mediaLista() vengono passate (tramite INDIRIZZO) alla funzione `calcola()` che eseguira` su di loro dei side effect, in modo che al ritorno da questa chiamata di `calcola()` le due variabili locali di mediaLista() contengano i valori della somma complessiva dei nodi della lista e del numero di nodi in lista, necessari per calcolare la media

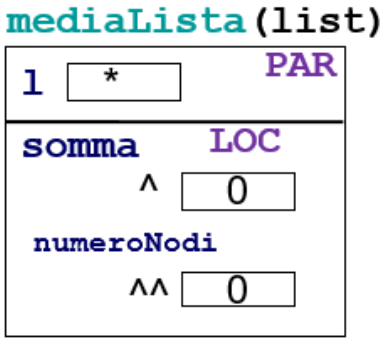
```
float mediaLista (TipoLista l) {

   int somma=0, numeroNodi=0;

  calcola (l, &somma, &numeroNodi);
  if (numeroNodi)
     return (float)somma / numeroNodi;
  else return 0.0;
}
```

se numeroNodi e` diverso da zero …

NB come noto, la media e` un double (o float) per ottenerlo da due interi bisogna che almeno una delle due sottoespressioni della divisione sia un double (o float)

Calcolo media elementi in lista (2/3)

`calcola(l,&somma,&numeroNodi)` viene richiamata da mediaLista() per assegnare somma e numeroNodi e permettere il calcolo della media

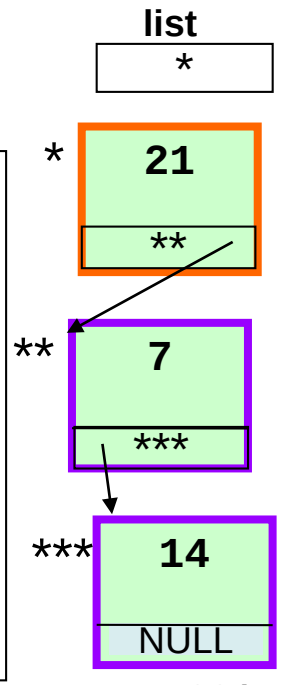



```
calcola(L, S, N) =
```

side effect

```
     = { se L NON VUOTA  - aggiungi primo a S;
                         - incrementa N
                         - calcola(LISTARESTO, S, N)
       { senno`          nulla
```

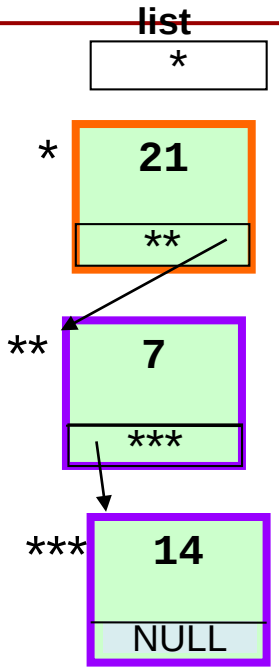
```
void calcola (TipoLista l, int *pS, int *pN) {
  if (l) {
    *pS += l->info;
    *pN += 1;
    calcola(l->next, pS, pN);
  }
return;
}
```



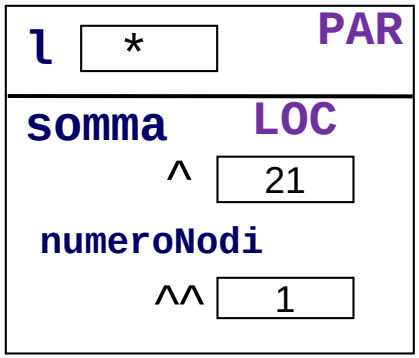
Calcolo media elementi in lista (3/3)

```
void calcola (TipoLista l, int *pS, int *pN) {
  if (l) {
    *pS += l->info;
    *pN += 1;
    calcola(l->next, pS, pN);
  }
return;}
```

vedi commenti alla fine



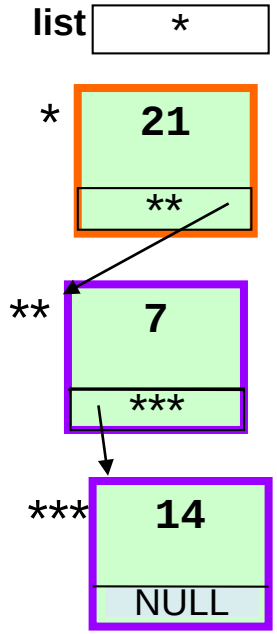
mediaLista(list)

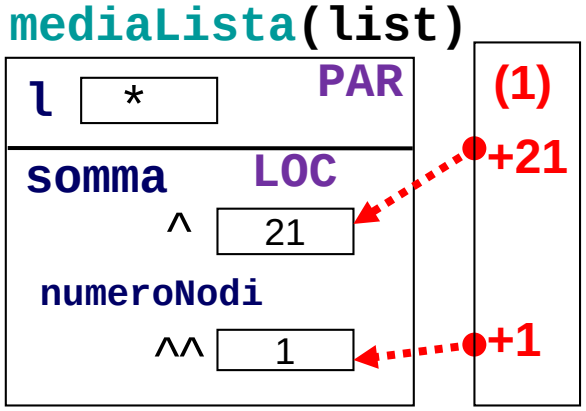


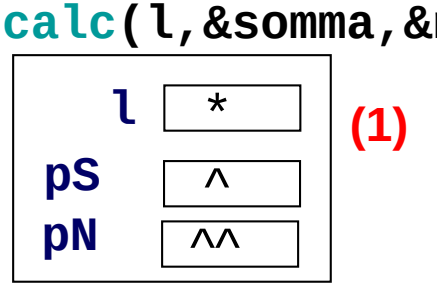
Calcolo media elementi in lista (3/3)

```
void calcola (TipoLista l, int *pS, int *pN) {
  if (l) {
    *pS += l->info;
    *pN += 1;
    calcola(l->next, pS, pN);
  }
return;}
```

vedi commenti alla fine



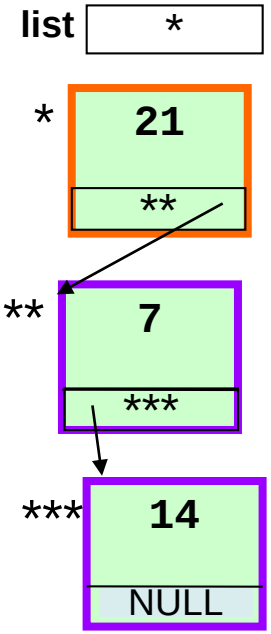


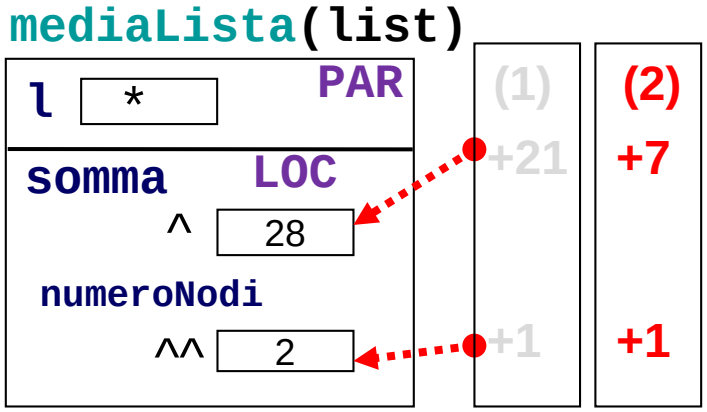


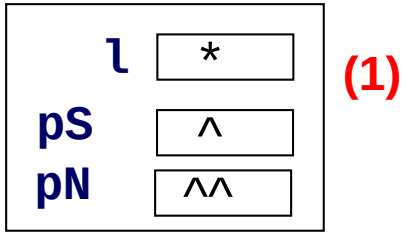
Calcolo media elementi in lista (3/3)

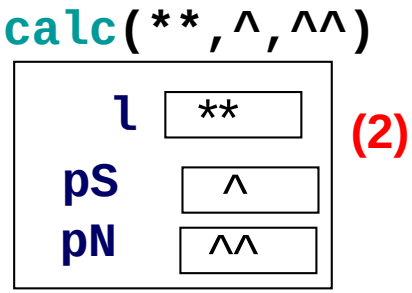
```
void calcola (TipoLista l, int *pS, int *pN) {
  if (l) {
    *pS += l->info;
    *pN += 1;
    calcola(l->next, pS, pN);
  }
return;}
```

vedi commenti alla fine





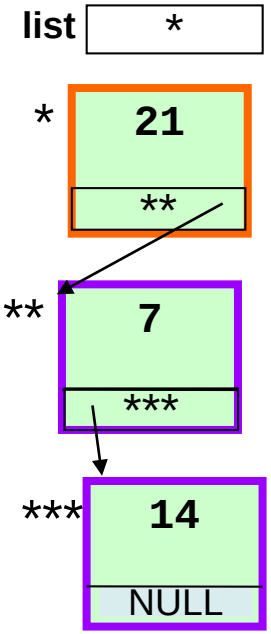


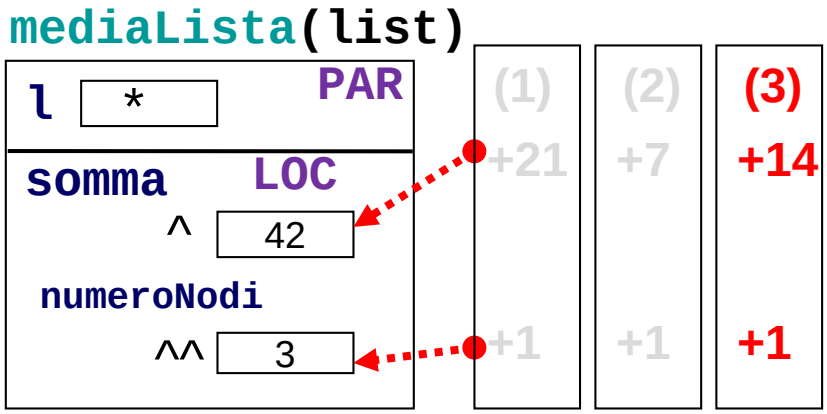


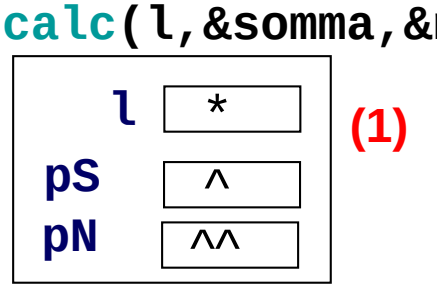
Calcolo media elementi in lista (3/3)

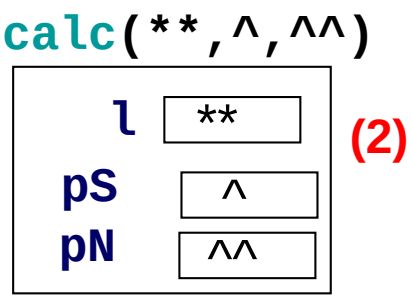
```
void calcola (TipoLista l, int *pS, int *pN) {
  if (l) {
    *pS += l->info;
    *pN += 1;
    calcola(l->next, pS, pN);
  }
return;}
```

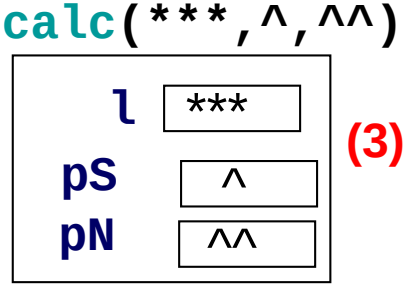
vedi commenti alla fine







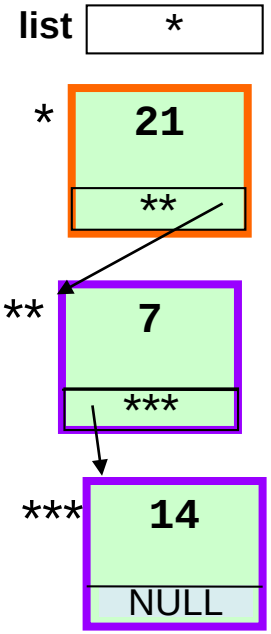


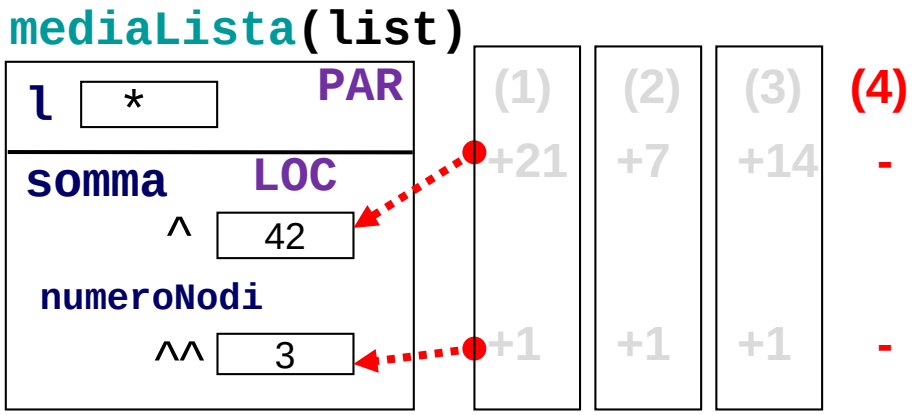


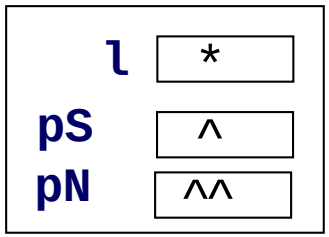
Calcolo media elementi in lista (3/3)

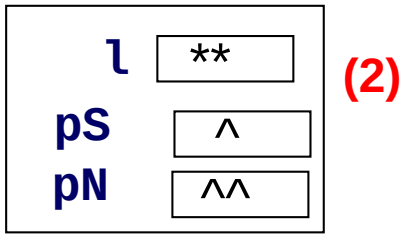
```
void calcola (TipoLista l, int *pS, int *pN) {
  if (l) {
    *pS += l->info;
    *pN += 1;
    calcola(l->next, pS, pN);
  }
return;}
```

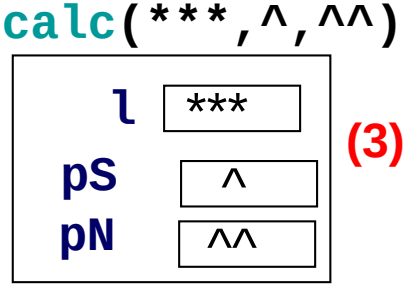
vedi commenti nella prossima slide

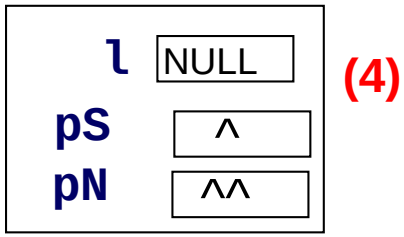








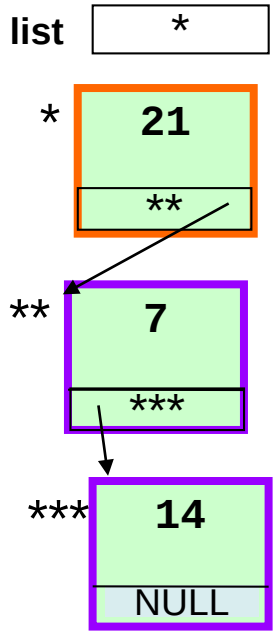


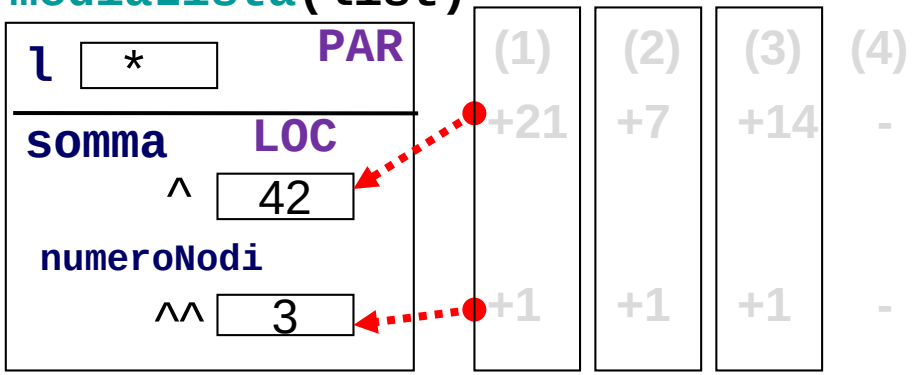


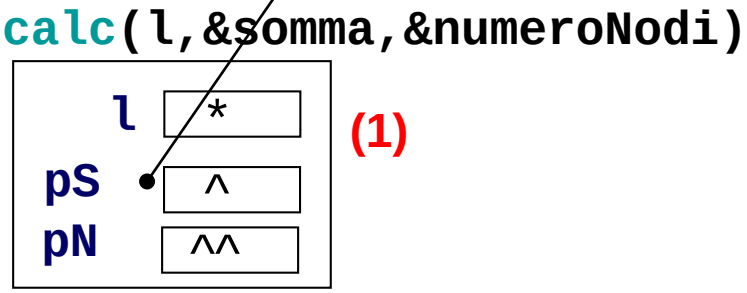
Calcolo media elementi in lista (3/3)

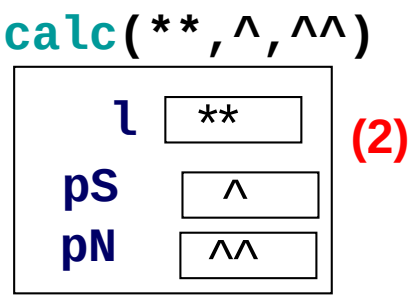
```
void calcola (TipoLista l, int *pS, int *pN) {
  if (l) {
    *pS += l->info;
    *pN += 1;
    calcola(l->next, pS, pN);
  }
return;}
```

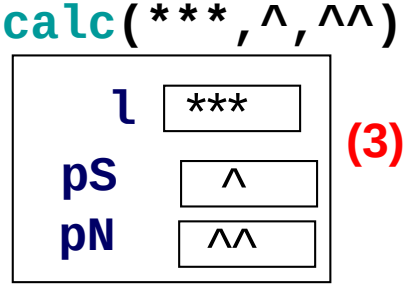
NB pS e` il medesimo indirizzo ricevuto dalla funzione; si tratta dell'indirizzo di somma (locazione appartenente a mediaLista).
Se questa chiamata di calcola avesse &pS come parametro attuale, sarebbe sbagliato perche' &pS e` l'indirizzo della locazione pS nel RDA attivo.
Se questa chiamata di calcola avesse *pS come parametro attuale, sarebbe sbagliato, perche' *pS e` il contenuto della locazione puntata da pS, cioe` il valore di somma (0 nella prima chiamata.)

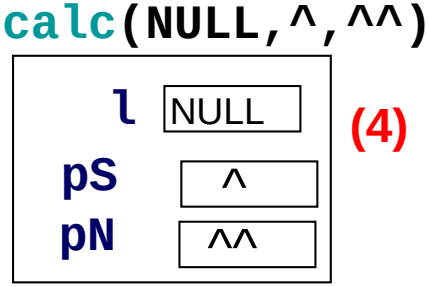








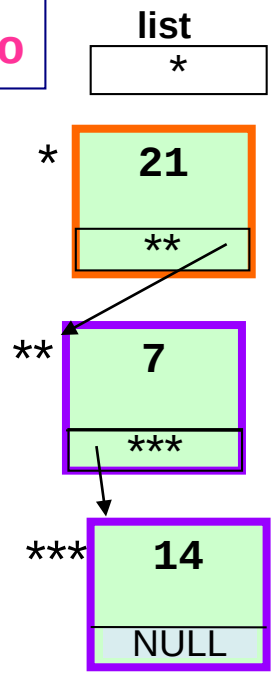




Eliminazione ricorsiva (1/3)

N.B. side effect su punt. iniziale
o sul punt. iniziale di una lista resto

```
elimina(ELEM, L)

     { se LISTA NON VUOTA { se 1º da elim.
=    {                    { else
     { LISTA VUOTA          NULLA
```

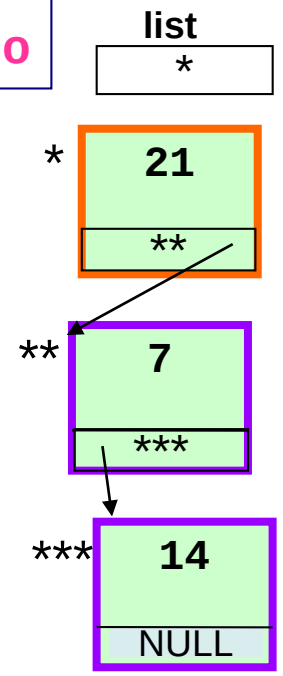


Eliminazione ricorsiva (1/3)

N.B. side effect su punt. iniziale
o sul punt. iniziale di una lista resto

```
elimina(ELEM, L)

        { se LISTA NON VUOTA { se 1º da elim.   elimina 1º
=       {                    { else
        { LISTA VUOTA          NULLA
```

side effect
(su list – cioe` su *plis – o su un campo next ...)



CHIAMATA: eliminaListaRic(val, &list)

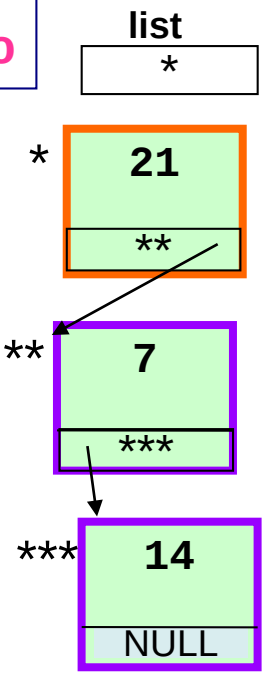
Eliminazione ricorsiva (1/3)

N.B. side effect su punt. iniziale
o sul punt. iniziale di una lista resto

```
elimina(ELEM, L)
```

=	se LISTA NON VUOTA	se 1º da elim. elimina 1º
		else elimina(ELEM, LISTARESTO)
	LISTA VUOTA	NULLA

side effect
(su list – cioe` su *plis – o su un campo next ...)

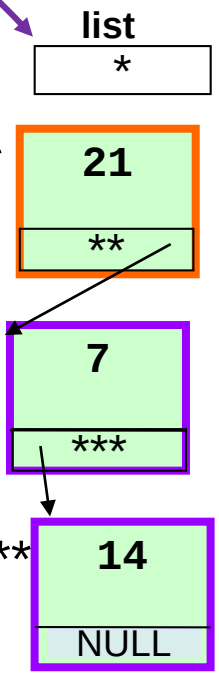


CHIAMATA: eliminaListaRic(val, &list)

Eliminazione ricorsiva (1/3)

N.B. side effect su punt. iniziale
o sul punt. iniziale di una lista resto

```
elimina(ELEM, L)

        { se LISTA NON VUOTA { se 1º da elim.   elimina 1º
=       {                    { else elimina(ELEM, LISTARESTO)
        { LISTA VUOTA          NULLA
```



side effect
(su list – cioe` su *plis – o su un campo next ...)

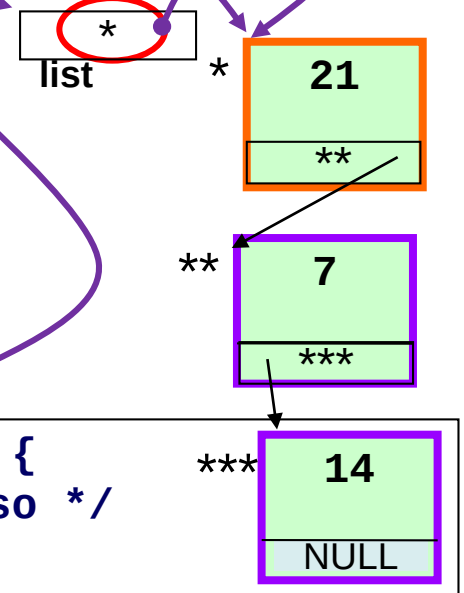
CHIAMATA: eliminaListaRic(val, &list)

```
void eliminaListaRic (TipoElem elem, TipoLista *plis) {

 TipoNodo * pprimo;   /* per eliminare il primo nel caso */

 if (*plis) {         /* lista non vuota */
    if ( uguali((*plis)->info, elem) ) { /* primo da elim */
        …}
…
```

Eliminazione ricorsiva (2/3)

N.B. side effect su punt. iniziale o sul punt. iniziale di una lista resto

```
elimina(ELEM, L)
        = { se LISTA NON VUOTA { se 1º da elim.  elimina 1º
                               { else elimina(ELEM, LISTARESTO)
          { LISTA VUOTA        NULLA
```




```
void eliminaListaRic (TipoElem elem, TipoLista *plis) {
 TipoNodo * pprimo;  /* per eliminare il primo nel caso */

 if (*plis) {        /* lista non vuota */
    if ( uguali((*plis)->info, elem) ) {
      pprimo = *plis;
     *plis = (*plis)->next;
      free(pprimo);
    }
    else eliminaListRic(elem, &((*plis)->next)) );
return;
}
```

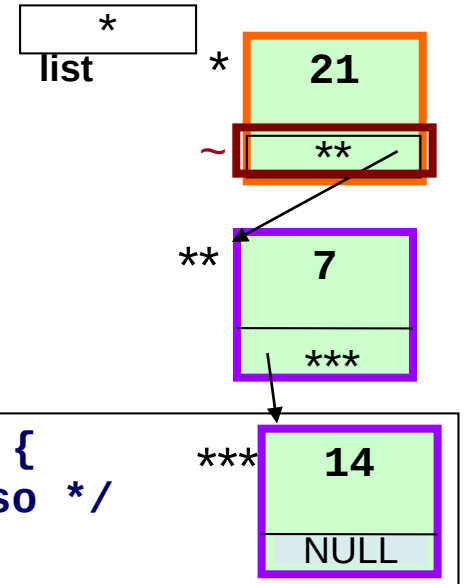
vedi commenti nella slide dopo

Eliminazione ricorsiva (2/3)

N.B. side effect su punt. iniziale o sul punt. iniziale di una lista resto

```
elimina(ELEM, L)
        = { se LISTA NON VUOTA : { se 1º da elim.  elimina 1º
                                 { else elimina(ELEM, LISTARESTO)
          { LISTA VUOTA        : NULLA
```

side effect

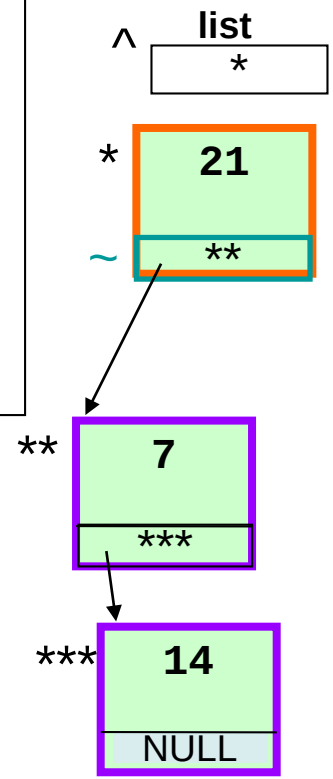



```
void eliminaListaRic (TipoElem elem, TipoLista *plis) {
 TipoNodo * pprimo;  /* per eliminare il primo nel caso */

 if (*plis) {        /* lista non vuota */
    if ( uguali((*plis)->info, elem) ) {
      pprimo = *plis;
     *plis = (*plis)->next;
      free(pprimo);
    }
    else eliminaListRic(elem, &((*plis)->next) );
 return;
 }
```

ad esempio:
*plis e` il puntatore al nodo 21;
allora `(*plis)->next` e` la locazione di indirizzo ~ e noi passiamo l'indirizzo di questa locazione (~) alla prossima chiamata di elimina, cioe` `&(*plis)->next`

Eliminazione ricorsiva (3/3)

```
void eliminaListaRic (TipoElem elem, TipoLista *plis) {
 TipoNodo * pprimo;  /* per eliminare il primo nel caso */

 if (*plis) {        /* lista non vuota */
    if ( uguali((*plis)->info, elem) ) {
      pprimo = *plis;
     *plis = (*plis)->next;
     free(pprimo);
    }
    else eliminaListRic(elem, &((*plis)->next) )
return;}
```



elimina(7,&list)

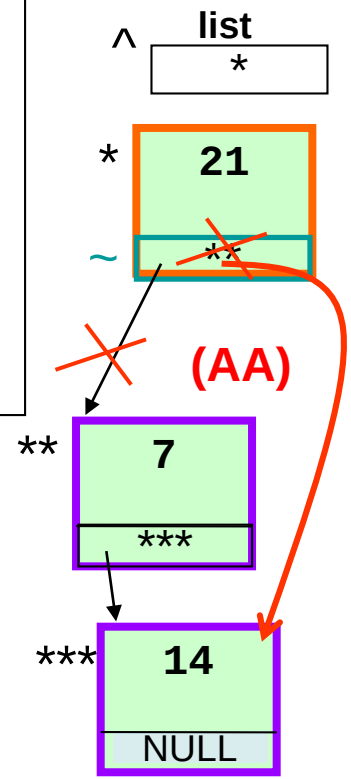
elem	7
plis	^
pprimo	
(*plis)->info e` 21	
&((*plis)->next) e` ~	

elimina(7, ~)

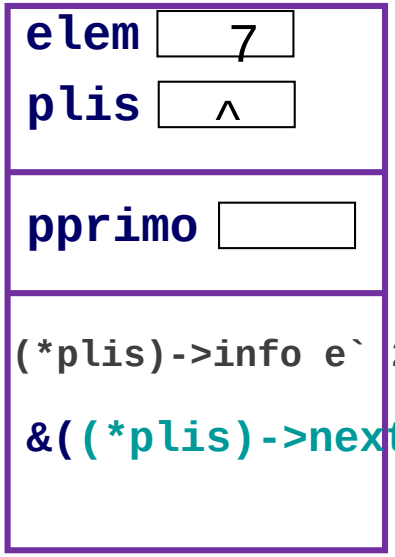
elem	7
plis	~
pprimo	**
(*plis)->info e` 7	
*plis = (*plis)->next; (AA)	

Eliminazione ricorsiva (3/3)

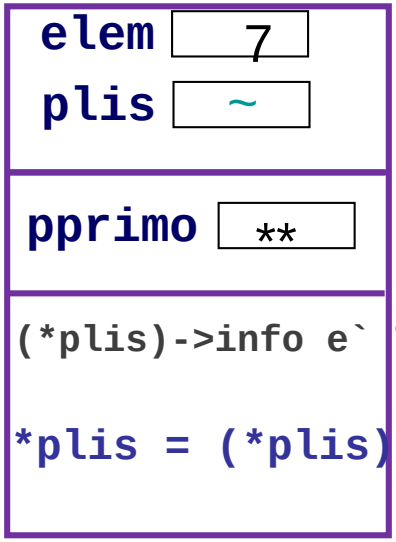
```
void eliminaListaRic (TipoElem elem, TipoLista *plis) {
 TipoNodo * pprimo;  /* per eliminare il primo nel caso */

 if (*plis) {        /* lista non vuota */
    if ( uguali((*plis)->info, elem) ) {
      pprimo = *plis;
     *plis = (*plis)->next;
     free(pprimo);
    }
    else eliminaListRic(elem, &((*plis)->next) )
return;}
```



elimina(7,&list)



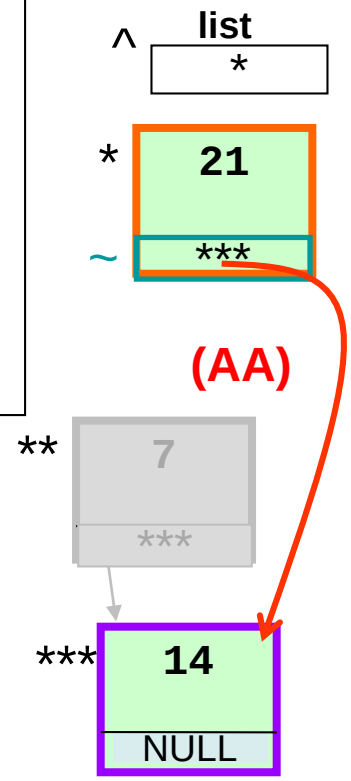
elimina(7, ~)



(AA)

Eliminazione ricorsiva (3/3)

```
void eliminaListaRic (TipoElem elem, TipoLista *plis) {
 TipoNodo * pprimo;  /* per eliminare il primo nel caso */

 if (*plis) {        /* lista non vuota */
    if ( uguali((*plis)->info, elem) ) {
      pprimo = *plis;
     *plis = (*plis)->next;
     free(pprimo);
    }
    else eliminaListRic(elem, &((*plis)->next) )
return;}
```



elimina(7,&list)

elem	7
plis	^
pprimo	
(*plis)->info e` 21	
&((*plis)->next) e` ~	

elimina(7, ~)

elem	7
plis	~
pprimo	**
(*plis)->info e` 7	
*plis = (*plis)->next; (AA)	
free(pprimo)	