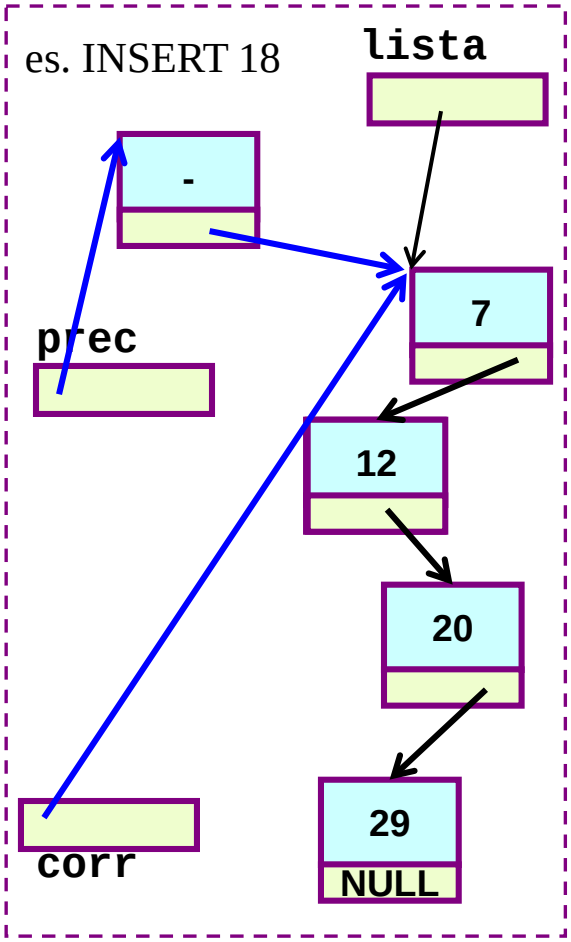
Previously on TdP, inserimento ordinato in lista



idea

- scansione con corr e prec

- alla fine *corr e` il primo nodo successore di quel che sara` il nuovo nodo

- e *prec sara` il predecessore del nuovo nodo

-

algoritmo

0) …

1) RG; `prec = pgen`
`corr = lista`

2) mentre `corr<>NULL`
AND `corr->info minore di elem`
far avanzare corr e prec "di conserva"

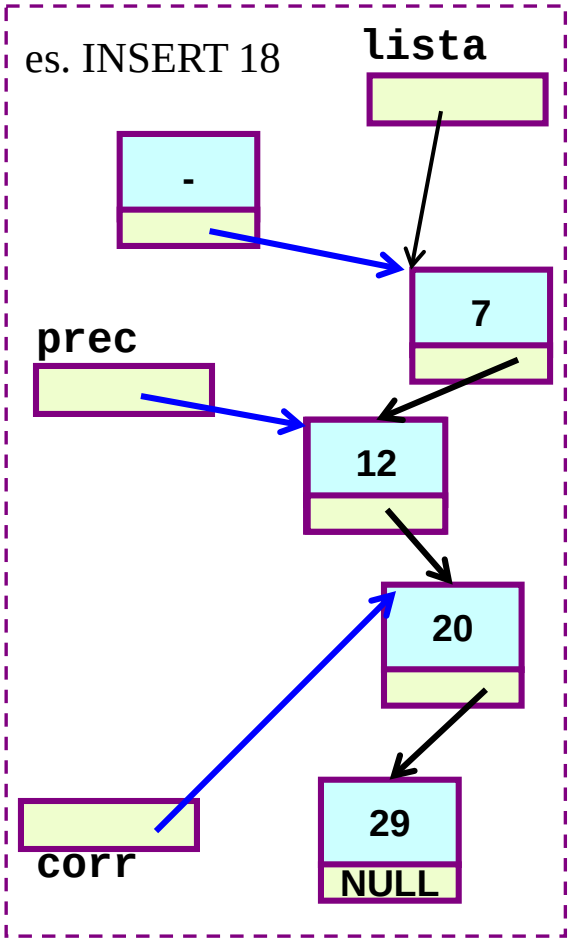
ORA *prec=predecessore e *corr=successore del nuovo nodo

3) creazione e inserimento nuovo nodo

4) riaggiustamenti finali (posizionamento radice lista e cancellazione RG)

```
lista = pgen->next
free(pgen)
```

Previously on TdP, inserimento ordinato in lista



idea

- scansione con corr e prec

- alla fine *corr e` il primo nodo successore di quel che sara` il nuovo nodo

- e *prec sara` il predecessore del nuovo nodo

- Ora bisogna creare il NUOVO NODO e INSERIRLO tra *prec e *corr

algoritmo

0) …

1) RG; `prec = pgen`
`corr = lista`

2) mentre `corr<>NULL`
AND `corr->info minore di elem`
far avanzare corr e prec "di conserva"

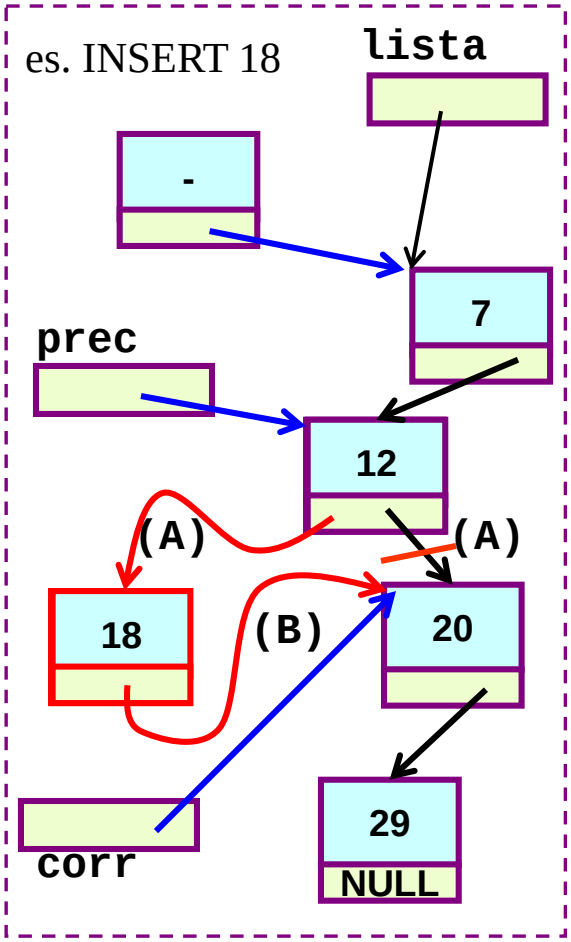
ORA *prec=predecessore e *corr=successore del nuovo nodo

3) creazione e inserimento nuovo nodo

4) riaggiustamenti finali (posizionamento radice lista e cancellazione RG)

```
lista = pgen->next
free(pgen)
```

Previously on TdP, inserimento ordinato in lista




```
3)

nuovo = malloc ...

nuovo->info = ...18

prec->next = nuovo (A)

nuovo->next = corr (B)
```

algoritmo

1) RG; `prec = pgen; corr=lista`

2) mentre `corr<>NULL AND corr->info minore di elem`
   avanzare corr e prec

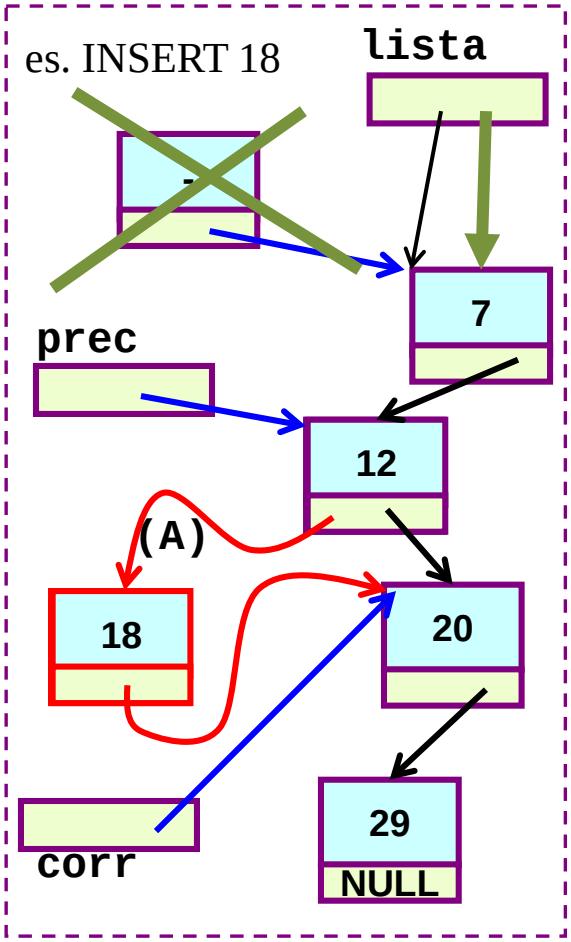
ORA *prec=predecessore e *corr=successore del nuovo nodo

3) creazione e inserimento nuovo nodo

4) riaggiustamenti finali (posizionamento radice lista e cancellazione RG)

```
lista = pgen->next
free(pgen)
```

Previously on TdP, inserimento ordinato in lista




```
4) riaggiustamenti
   finali

  lista = pgen->next

  free(pgen)
```

algoritmo

0) …

1) RG; `prec = pgen`
`corr = lista`

2) mentre `corr<>NULL`
`AND corr->info minore di elem`
far avanzare corr e prec "di conserva"

ORA *prec=predecessore e *corr=successore del nuovo nodo

3) creazione e inserimento nuovo nodo

4) riaggiustamenti finali (posizionamento radice lista e cancellazione RG)

```
lista = pgen->next
free(pgen)
```

E ora ... Lezione 25 (LISTE-4)

Rappresentazione concreta del Tipo di Dati Astratto LISTA.

- eliminazione di un elemento da lista
  - nella funzione main()
    - algoritmo
    - casi particolari
  - mediante funzione elimDaLista()

- inserimento in lista ordinata
  - nella funzione in esecuzione (ad esempio la main())
  - **mediante funzione insOrdLista()**
  - **versione alternativa: con record generatore ma senza corr**

- **eliminazione di un elemento da lista**
  - **versione alternativa: senza record generatore - elimDaLista2()**

```
...typedef int TipoElem;...
... typedef TipoNodo * PuntNodoLista

int main() {
 TipoLista lista;
 TipoElem elem;
...
  /* costruzione lista */
...
  leggiElem(&elem);  /* l'el. da aggiungere */
...
  risposta =
  insOrdLista(&lista, elem); /* inserimento ordinato */
...
  stampaLista(lista);
...
...
return 0;
}
```

Inserimento ordinato in lista - con funzione di inserimento, premessa sui tipi

```
...typedef int TipoElem;...
... typedef TipoNodo * PuntNodoLista

int main() {
 TipoLista lista;
 TipoElem elem;
...
  /* costruzione lista */
...
  leggiElem(&elem);  /* l'el. da eliminare */
...
  risposta =
  insOrdLista(&lista, elem); /* inserimento ordinato */
...
  stampaLista(lista);
...
...
return 0;
}
```

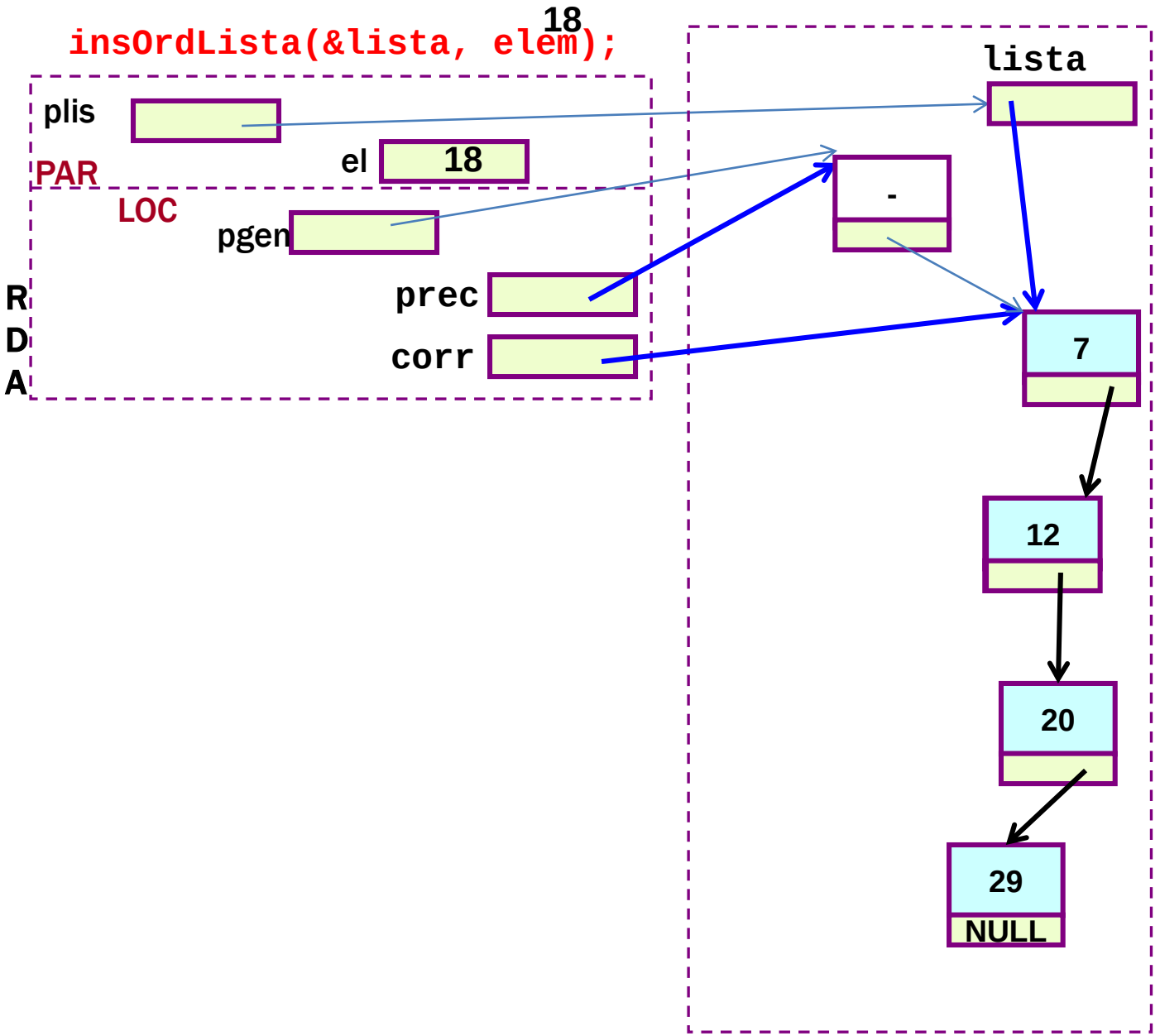
PuntNodoLista e` il tipo dei puntatori usati come appoggio per gestire le liste: servono a puntare a singoli nodi, ad esempio durante la scansione della lista o per la creazione di nuovi nodi - lo usiamo qui perche' e` usato un bel po' negli esercizi forniti nella directory pubblica

risposta conterra` 1/0 dipendentemente dal successo o insuccesso dell'inserimento

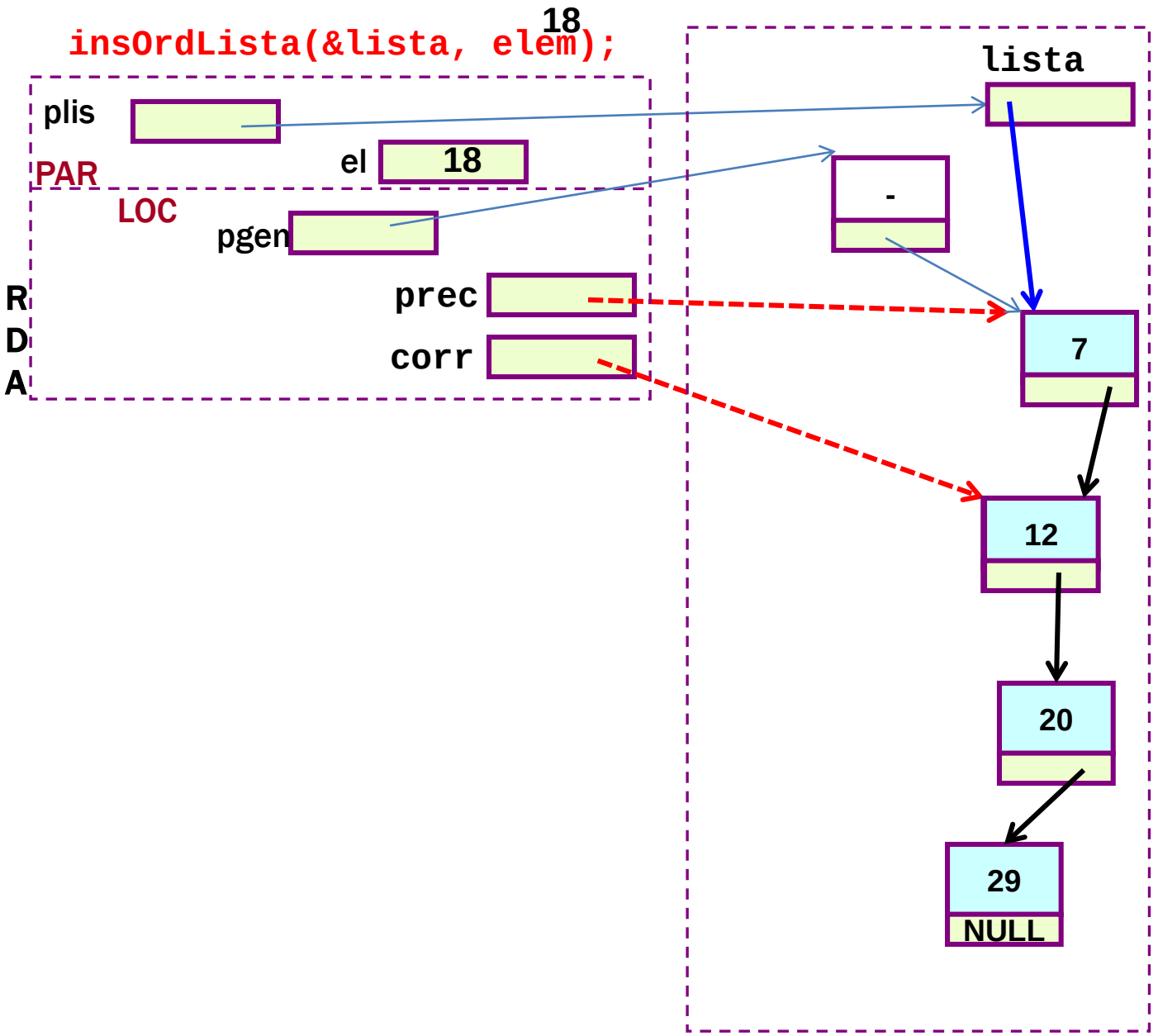
il programma, ad un certo punto gestisce l'inserimento ordinato di un elemento nella lista (elemento specificato in input)

NB il puntatore all'inizio della lista non cambia quasi mai, ma potrebbe cambiare (nel caso in cui il nodo da inserire in lista sia destinato ad essere il primo in lista) quindi dobbiamo gestire la funzione in modo che essa possa provocare un effetto collaterale

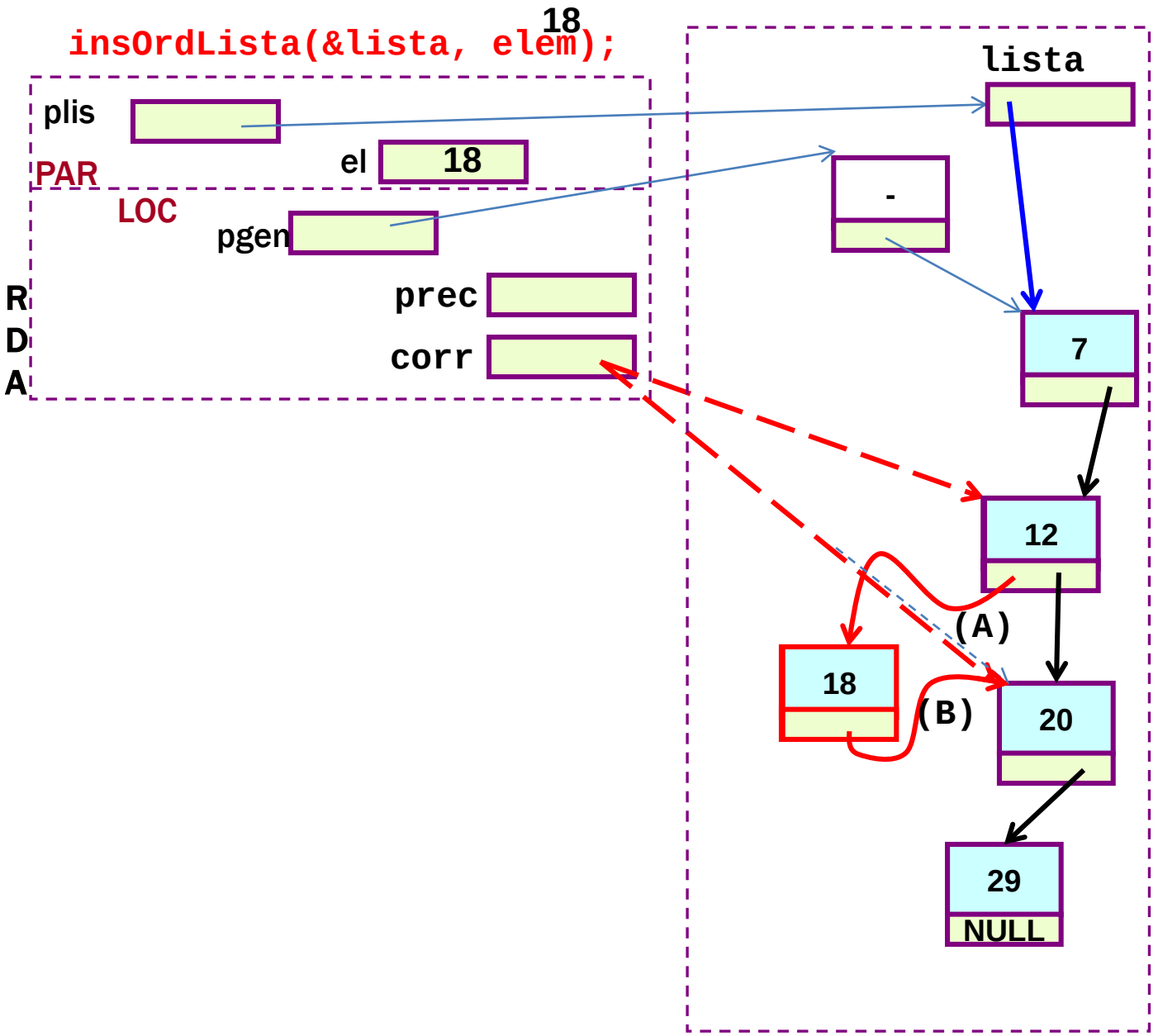
Inserimento ordinato in lista - uso di funzione di inserimento 1/5



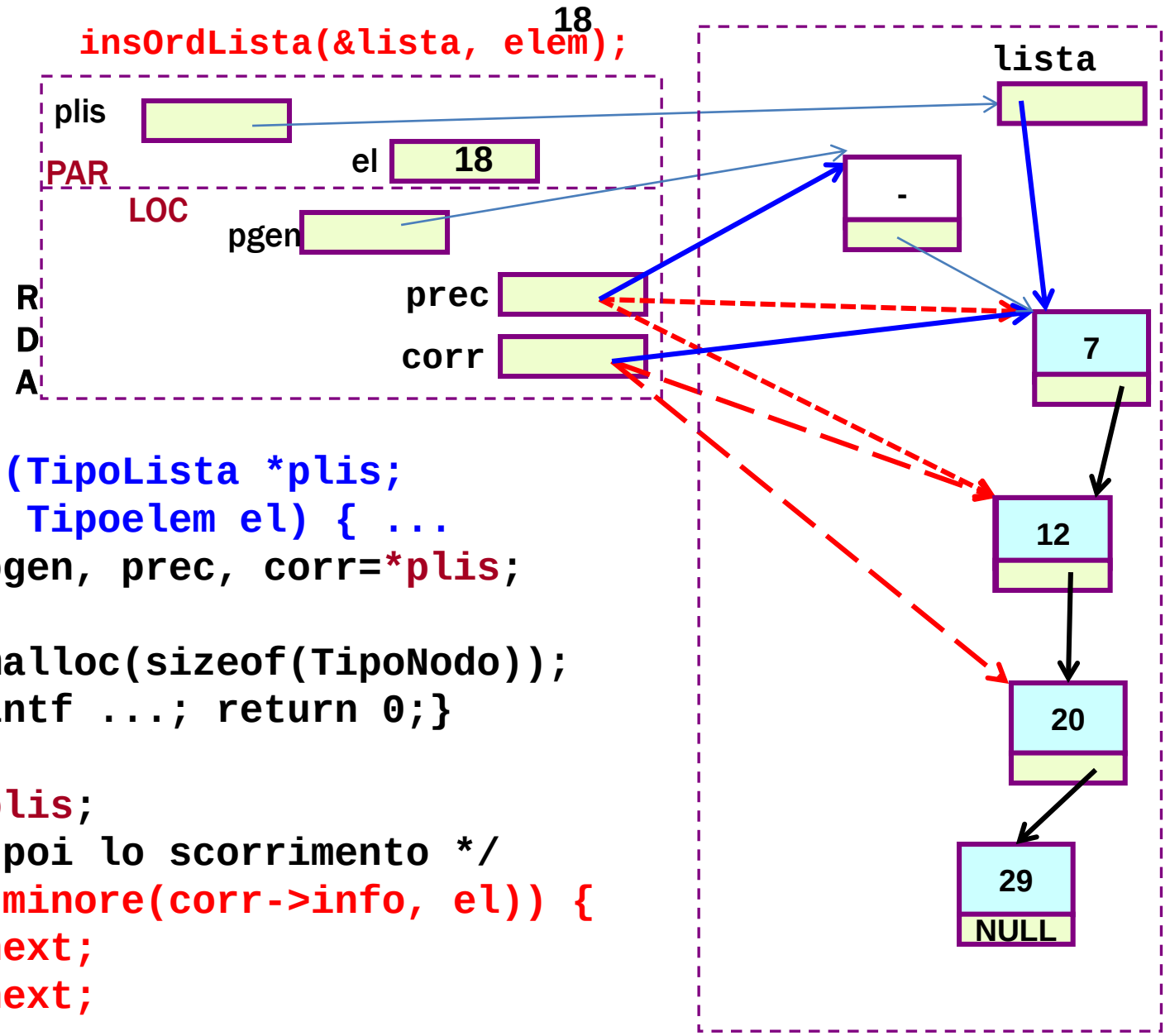
Inserimento ordinato in lista - uso di funzione di inserimento 1/5



Inserimento ordinato in lista - uso di funzione di inserimento 1/5



Inserimento ordinato in lista - uso di funzione di inserimento 1/5



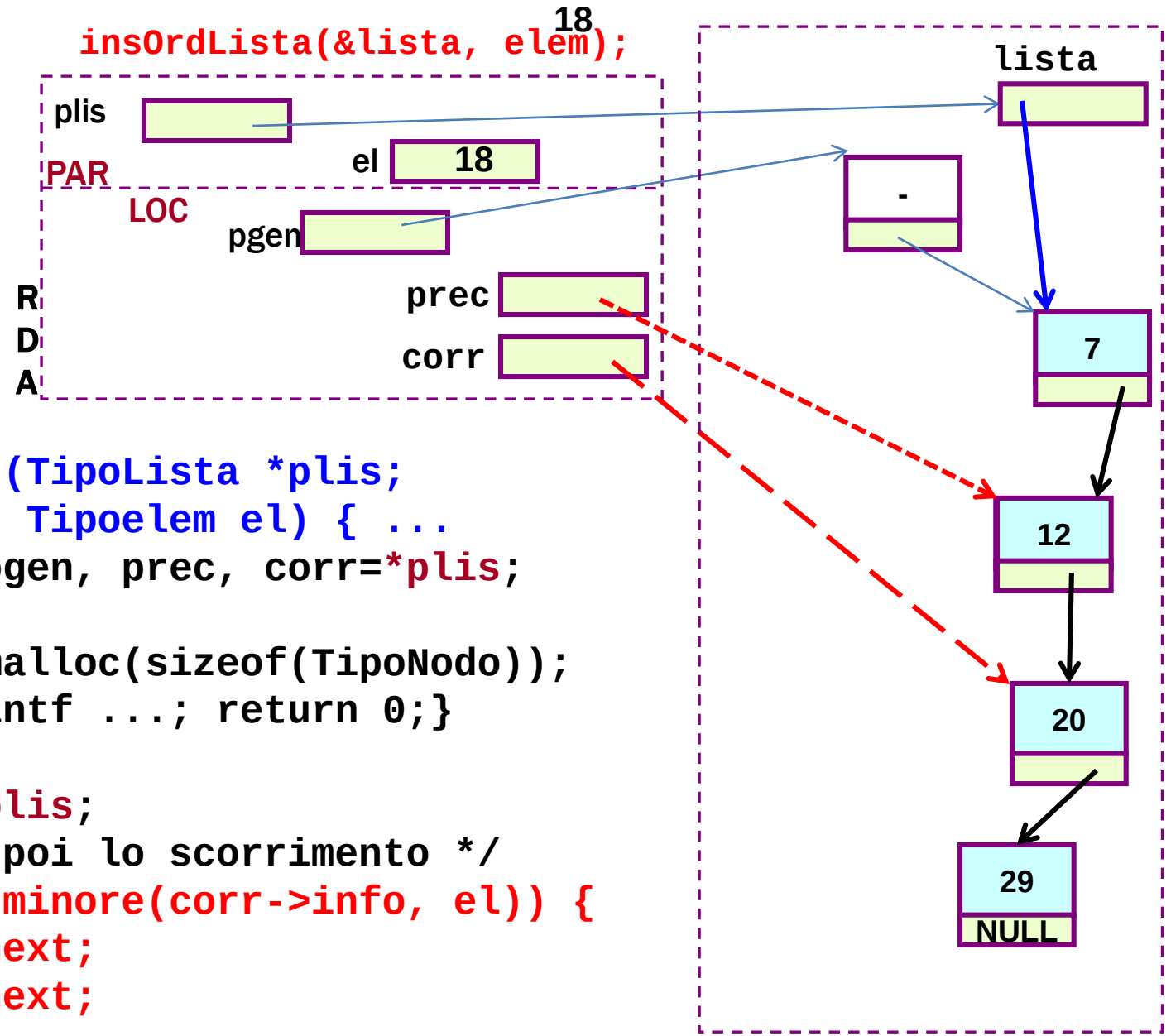

```
int insOrdLista (TipoLista *plis;
                 Tipoelem el) { ...
 PuntNodoLista pgen, prec, corr=*plis;

 prec = pgen = malloc(sizeof(TipoNodo));
 if (!pgen) {printf ...; return 0;}

 pgen->next = *plis;
              /* poi lo scorrimento */
 while (corr && minore(corr->info, el)) {
  corr = corr->next;
  prec = prec->next;
 }
```

Inserimento ordinato in lista - uso di funzione di inserimento 1/5

plis punta a lista e potra` permettere il side effect;

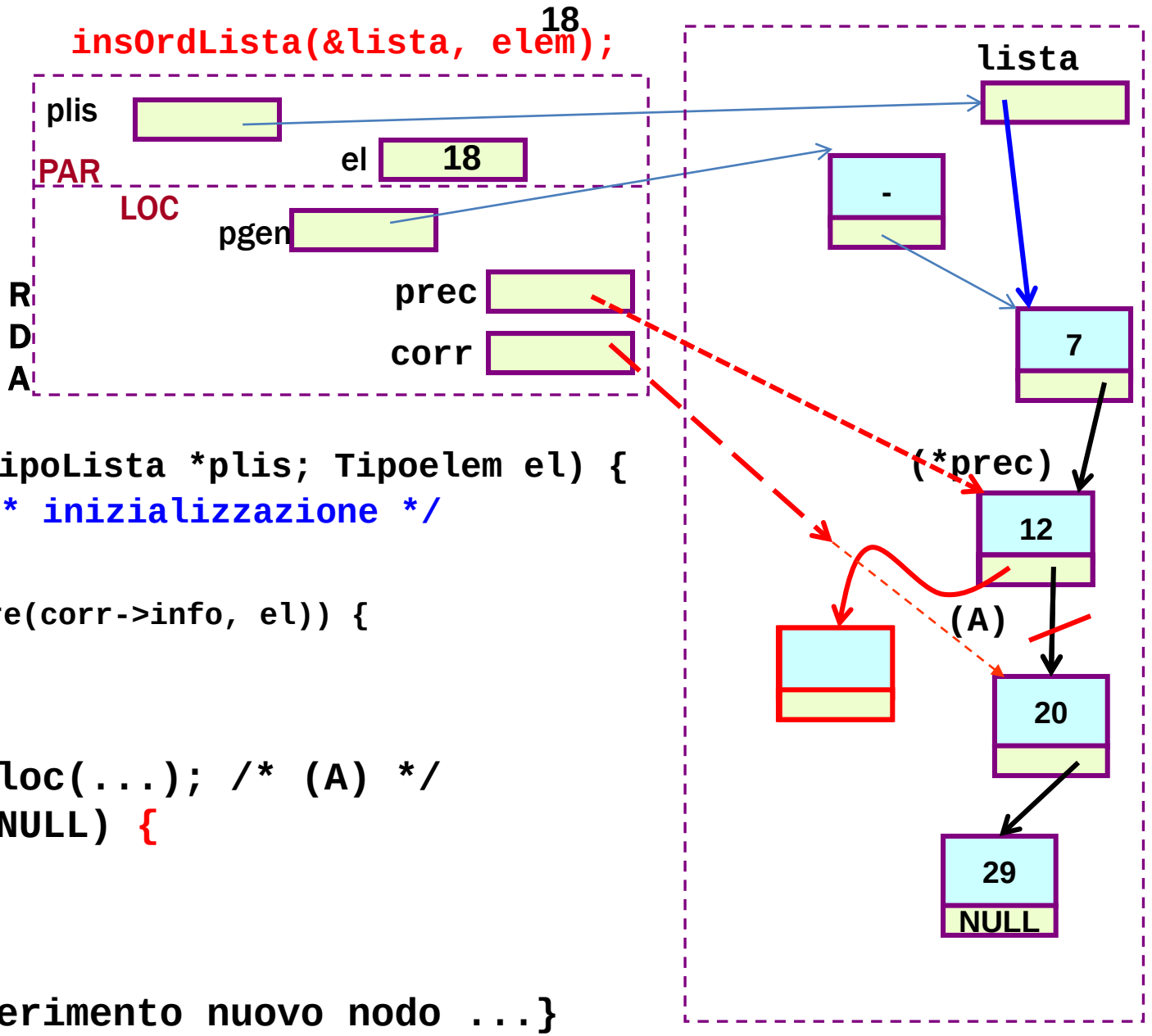
corr parte puntando sul 7 e finisce puntando sul 20.

prec parte puntando su *pgen e finisce puntando su 12;



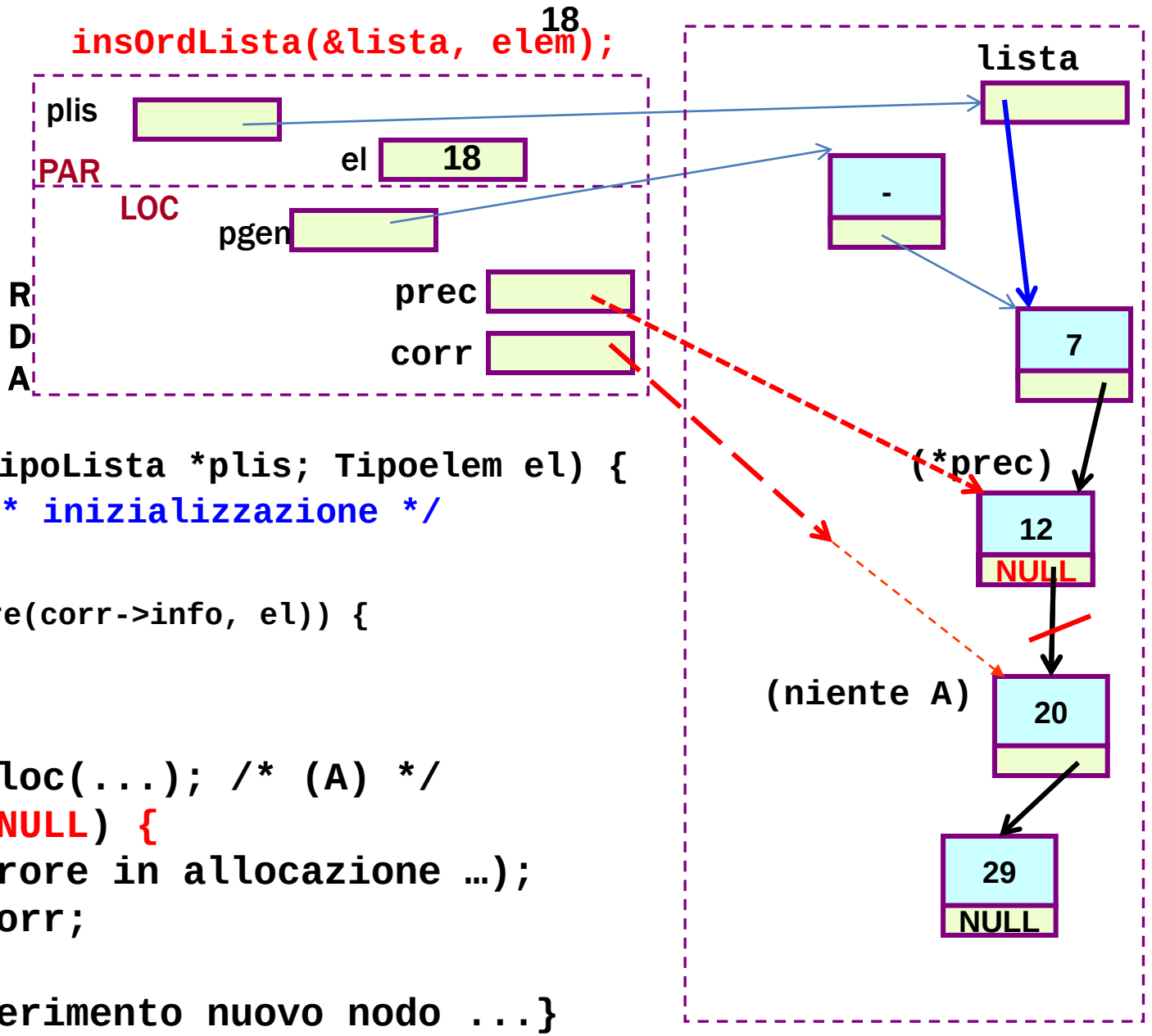

```
int insOrdLista (TipoLista *plis;
                 Tipoelem el) { ...
 PuntNodoLista pgen, prec, corr=*plis;

 prec = pgen = malloc(sizeof(TipoNodo));
 if (!pgen) {printf ...; return 0;}

 pgen->next = *plis;
              /* poi lo scorrimento */
 while (corr && minore(corr->info, el)) {
  corr = corr->next;
  prec = prec->next;
 }
```

Inserimento ordinato in lista - uso di funzione di inserimento 2/5




```
int insOrdLista (TipoLista *plis; Tipoelem el) {
... int ris = 1; /* inizializzazione */
... ...
 while (corr && minore(corr->info, el)) {
  corr = corr->next;
  prec = prec->next;
 }
prec->next = malloc(...); /* (A) */
if (prec->next==NULL) {



} else { ... inserimento nuovo nodo ...}
```

Inserimento ordinato in lista - uso di funzione di inserimento 2/5

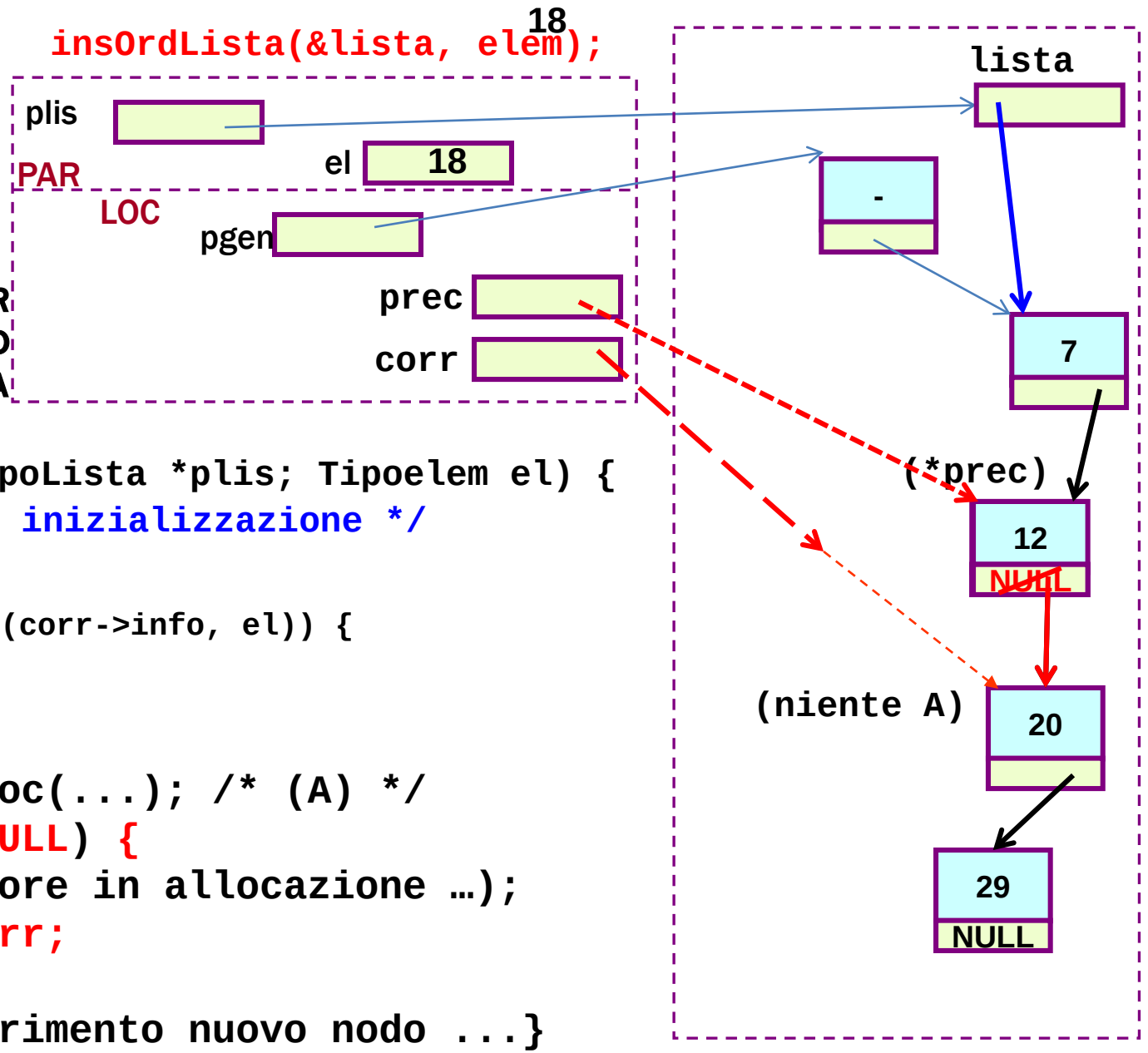



```
int insOrdLista (TipoLista *plis; Tipoelem el) {
... int ris = 1; /* inizializzazione */
... ...
 while (corr && minore(corr->info, el)) {
  corr = corr->next;
  prec = prec->next;
 }
prec->next = malloc(...); /* (A) */
if (prec->next==NULL) {
    printf( … errore in allocazione …);
    prec->next=corr;
    ris = 0;
} else { ... inserimento nuovo nodo ...}
```

Inserimento ordinato in lista - uso di funzione di inserimento 2/5

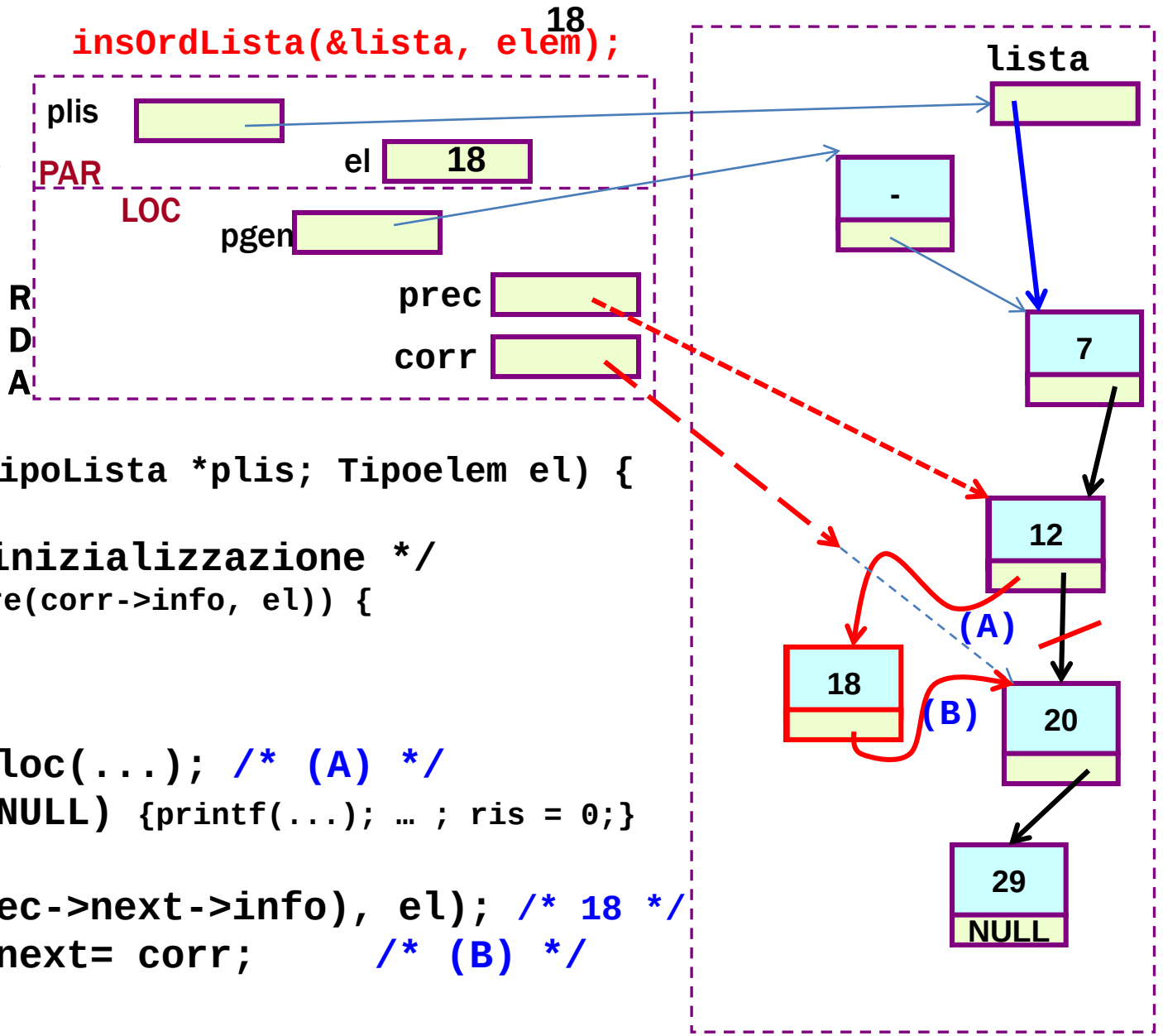
`prec->next` e` stato modificato con la malloc, quindi nel caso in cui la malloc non sia andata a buon fine deve essere ripristinato al valore preedente (`corr`).;
`ris` contiene il successo/insuccesso (1/0) dell'inserimento: se l'allocazione del nuovo nodo e` andata male, qui viene assegnato a 0; altrimenti si esegue l'inserimento del nuovo nodo. Se l'inserimento non avviene, si prosegue verso la deallocazione del RG, con ris==0, cosi` si puo` chiudere la funzione e restituire il giusto risultato

```
... poLista *plis; Tipoelem el) {
... int ris = 1; /* inizializzazione */
... ...
 while (corr && minore(corr->info, el)) {
  corr = corr->next;
  prec = prec->next;
 }
prec->next = malloc(...); /* (A) */
if (prec->next==NULL) {
    printf( … errore in allocazione …);
    prec->next=corr;
    ris = 0;
} else { ... inserimento nuovo nodo ...}
```



Inserimento ordinato in lista - uso di funzione di inserimento 3/5

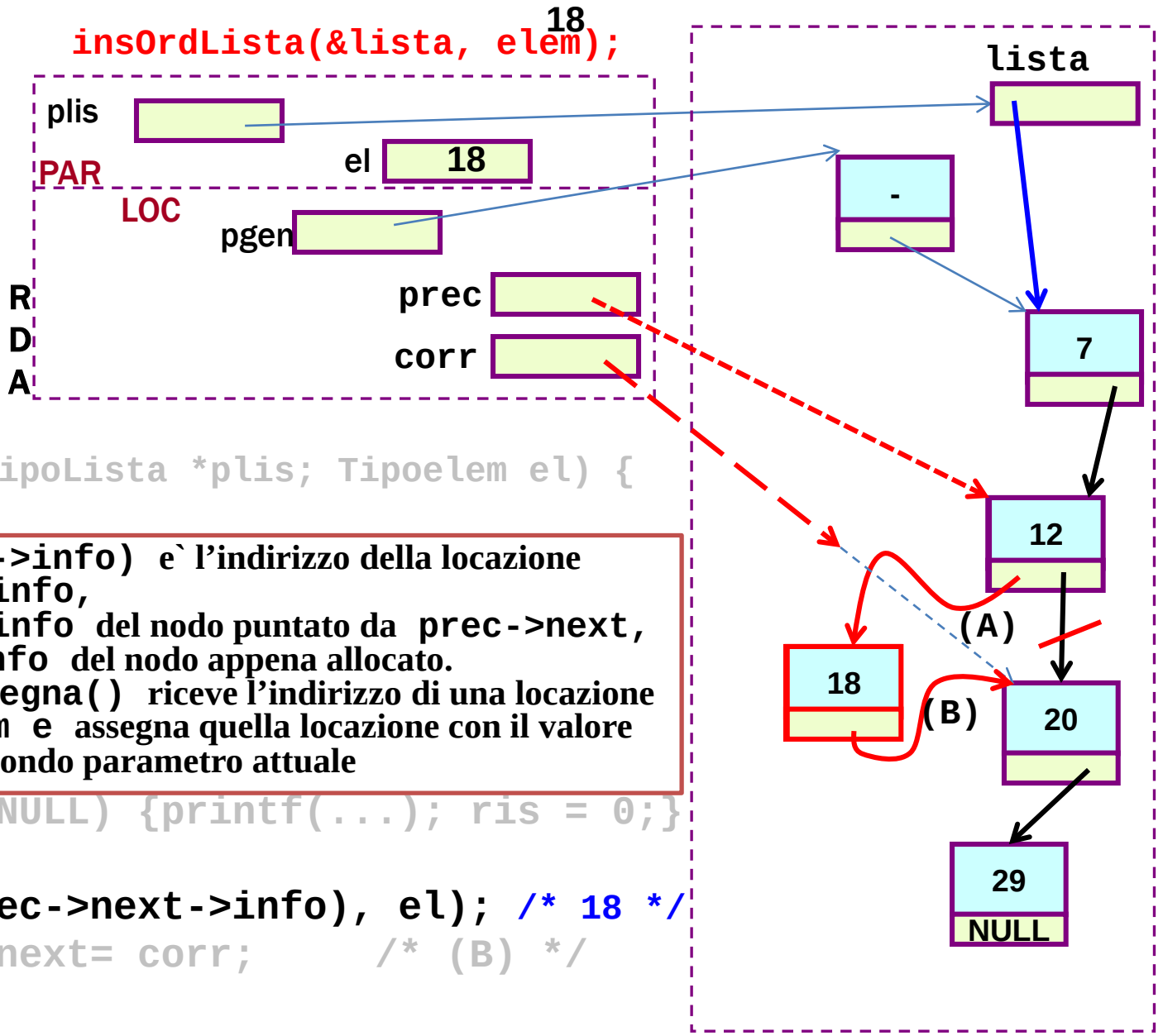
Torniamo a quando l'allocazione era andata a buon fine…




```
int insOrdLista (TipoLista *plis; Tipoelem el) {
...
 int ris = 1; /* inizializzazione */
 while (corr && minore(corr->info, el)) {
  corr = corr->next;
  prec = prec->next;
 }
prec->next = malloc(...); /* (A) */
if (prec->next==NULL) {printf(...); … ; ris = 0;}
else {
    assegna(&(prec->next->info), el); /* 18 */
    prec->next->next= corr;     /* (B) */
} …chiusura
```

Inserimento ordinato in lista - uso di funzione di inserimento 3/5

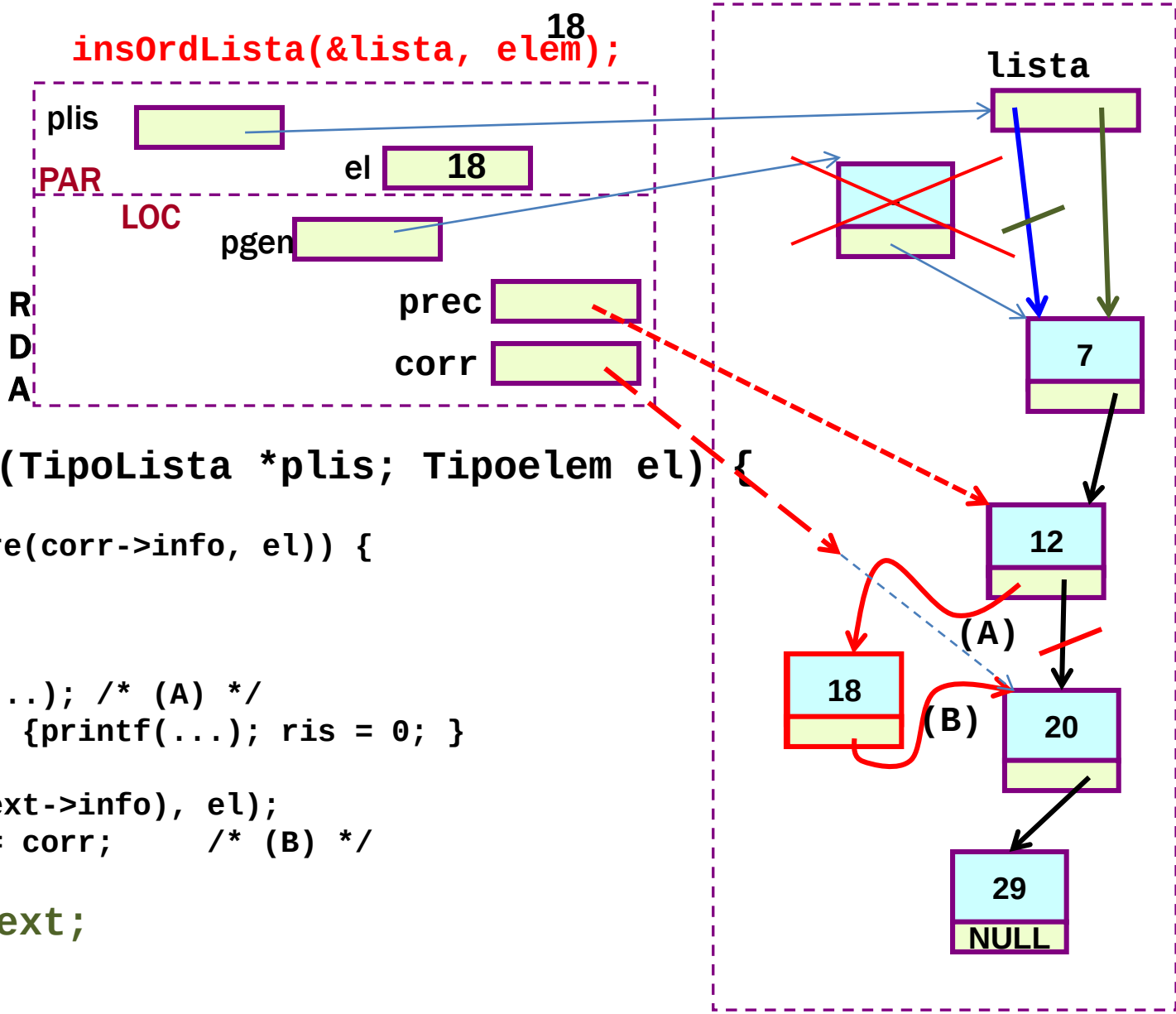
NB




```
int insOrdLista (TipoLista *plis; Tipoelem el) {
...
 int
 while
  corr
  prec
 }
prec
if (prec->next==NULL) {printf(...); ris = 0;}
else {
    assegna(&(prec->next->info), el); /* 18 */
    prec->next->next= corr;     /* (B) */
} ...chiusura
```

`&(prec->next->info)` e` l'indirizzo della locazione `prec->next->info`, cioe` del campo `info` del nodo puntato da `prec->next`, cioe` il campo `info` del nodo appena allocato.
La funzione `assegna()` riceve l'indirizzo di una locazione di tipo `TipoElem e` assegna quella locazione con il valore ricevuto come secondo parametro attuale

Inserimento ordinato in lista - uso di funzione di inserimento 4/5

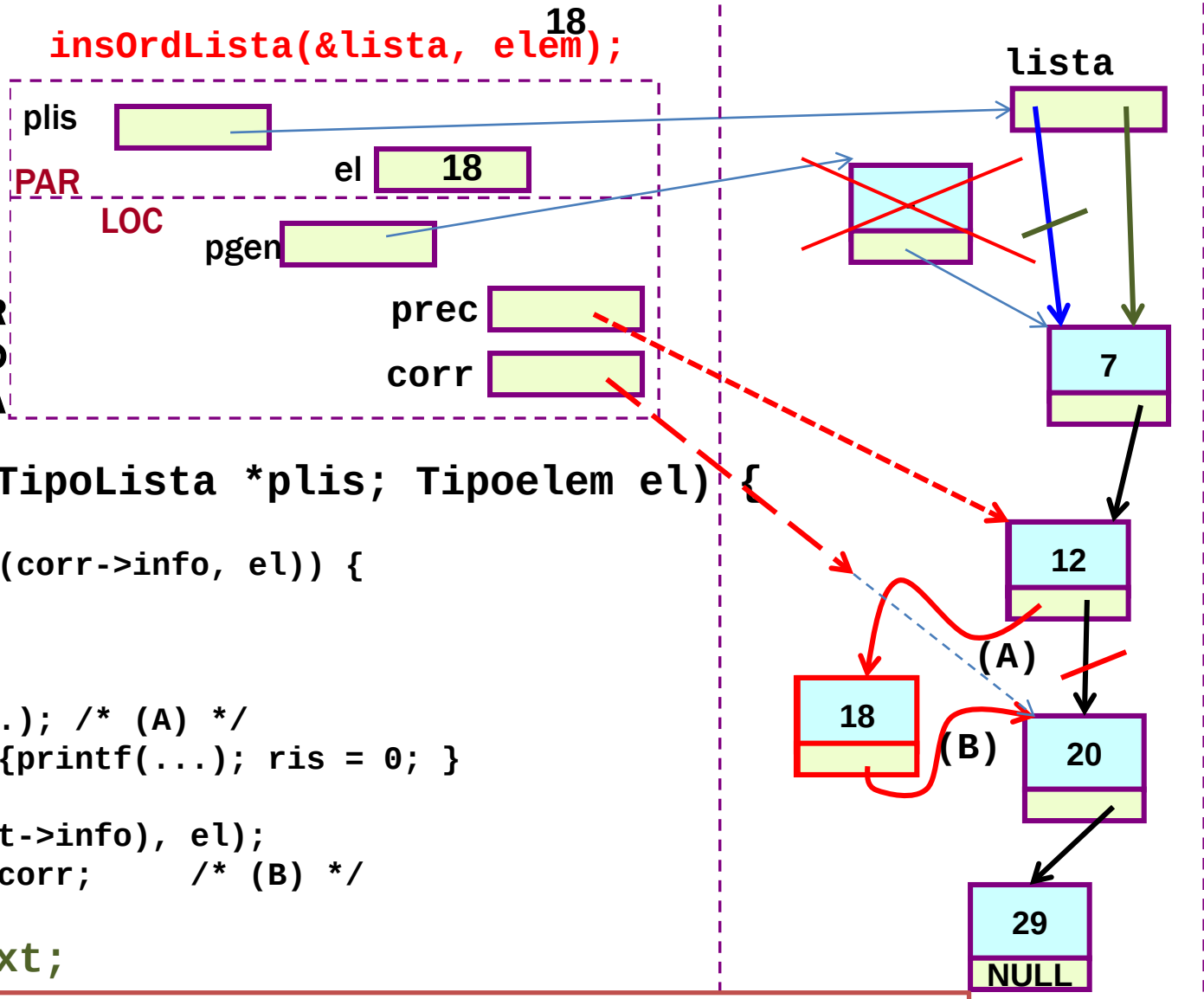



```
int insOrdLista (TipoLista *plis; Tipoelem el) {
...int ris = 1;
 while (corr && minore(corr->info, el)) {
  corr = corr->next;
  prec = prec->next;
 }
prec->next = malloc(...); /* (A) */
if (prec->next==NULL) {printf(...); ris = 0; }
else {
    assegna(&(prec-next->info), el);
    prec->next->next= corr;     /* (B) */
}
 *plis = pgen->next;
 free(pgen);
return ris; }
```

Inserimento ordinato in lista - uso di funzione di inserimento 4/5

`plis` punta a `lista` e potra` permettere il side effect;

chiusura dell'algoritmo: riposizionamento di lista (cioe` di *plis) in modo che punti comunque al primo el. della lista, ed eliminazione del RG.




```
int insOrdLista (TipoLista *plis; Tipoelem el) {
...int ris = 1;
 while (corr && minore(corr->info, el)) {
  corr = corr->next;
  prec = prec->next;
 }
prec->next = malloc(...); /* (A) */
if (prec->next==NULL) {printf(...); ris = 0; }
else {
    assegna(&(prec-next->info), el);
    prec->next->next= corr;     /* (B) */
}
 *plis = pgen->next;
 free(pgen);
return ris; }
```

se ris non e` stato riassegnato, vale ancora 1 (come da inizializzazione); Lo potremmo aver riassegnato (a zero) solo nel caso in cui il nuovo nodo non fosse stato allocato, il che indicherebbe insuccesso dell'operazione.

nel caso **typedef int TipoElem**

```
int  minore( Tipoelem elem1, Tipoelem elem2 ) {
 return (elem1 < elem2);
}
```

---

nel caso **typedef TipoVolo Tipoelem ??**

una soluzione potrebbe essere

```
int  minore( Tipoelem elem1, Tipoelem elem2 ) {
 if (strcmp(elem1.codice, elem2.codice) < 0)
   return 1;
 else return 0;
}
```

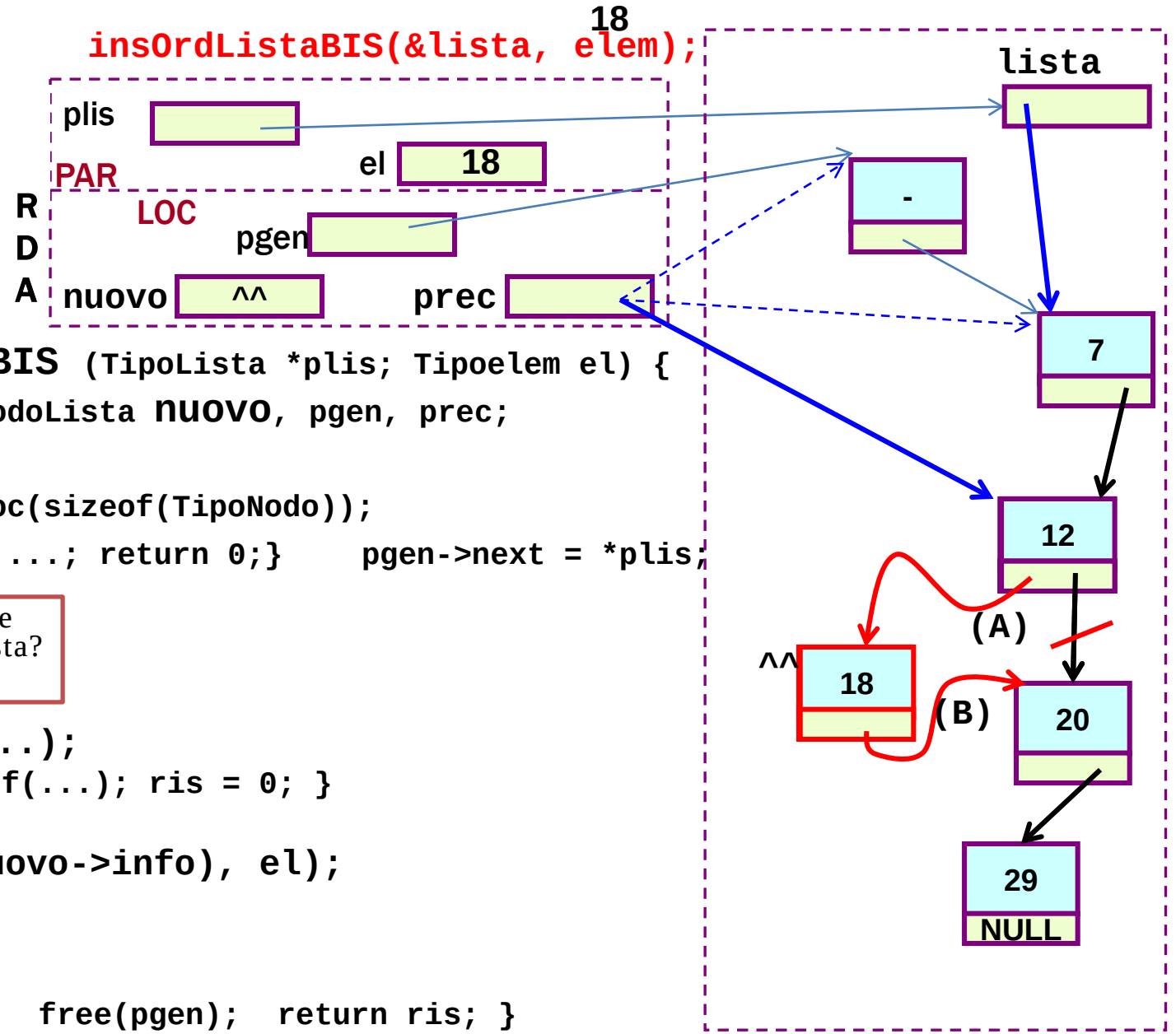
un'altra potrebbe essere

```
int  minore( Tipoelem elem1, Tipoelem elem2 ) {
 if (elem1.oraPartenza.ore < elem2.oraPartenza.ore))
   return 1;
 else return 0;
```

} ... Dipende da come intendiamo il concetto di *minore* per questo particolare TipoElem …

Inserimento ordinato in lista - uso di funzione di inserimento - alternativa 1/4

sempre con il RG, ma usando solo `prec` (niente `corr`)



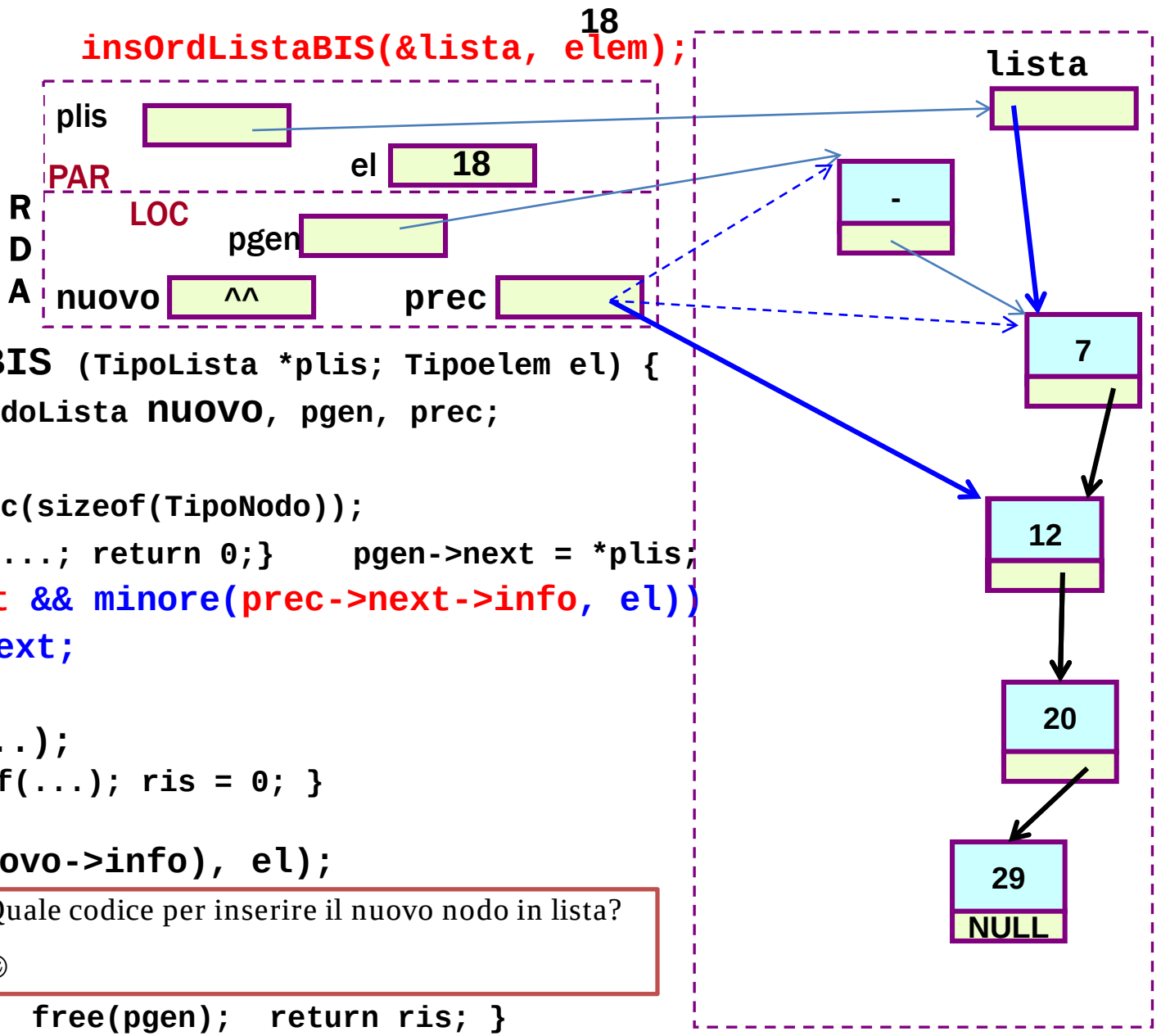

```
int insOrdListaBIS (TipoLista *plis; Tipoelem el) {
 int ris = 1; PuntNodoLista nuovo, pgen, prec;

 prec = pgen = malloc(sizeof(TipoNodo));
 if (!pgen) {printf ...; return 0;}    pgen->next = *plis;
```

quale codice per scandire fino a questo punto la lista? ☺

```
 nuovo = malloc(...);
  if (!nuovo) {printf(...); ris = 0; }
  else {
      assegna(&(nuovo->info), el);


}
 *plis = pgen->next;  free(pgen);  return ris; }
```

Inserimento ordinato in lista - uso di funzione di inserimento - alternativa 1/4

sempre con il RG, ma usando solo prec (niente corr)



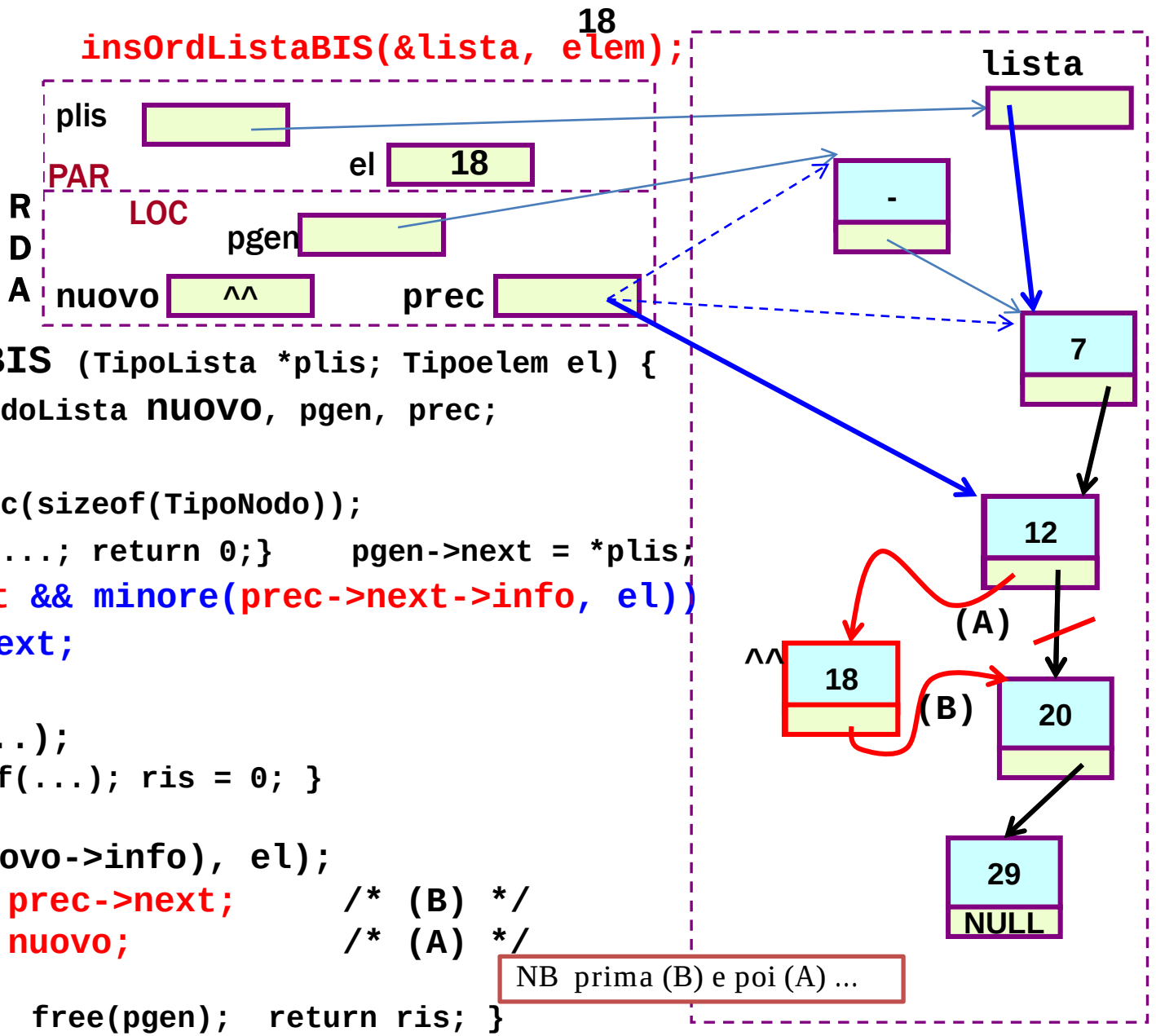

```
int insOrdListaBIS (TipoLista *plis; Tipoelem el) {
 int ris = 1; PuntNodoLista nuovo, pgen, prec;

 prec = pgen = malloc(sizeof(TipoNodo));
 if (!pgen) {printf ...; return 0;}    pgen->next = *plis;
 while (prec->next && minore(prec->next->info, el))
    prec = prec->next;

 nuovo = malloc(...);
  if (!nuovo) {printf(...); ris = 0; }
  else {
      assegna(&(nuovo->info), el);
}
 *plis = pgen->next;  free(pgen);  return ris; }
```

Quale codice per inserire il nuovo nodo in lista?

☺

Inserimento ordinato in lista - uso di funzione di inserimento - alternativa 1/4

sempre con il RG, ma usando solo prec (niente corr)



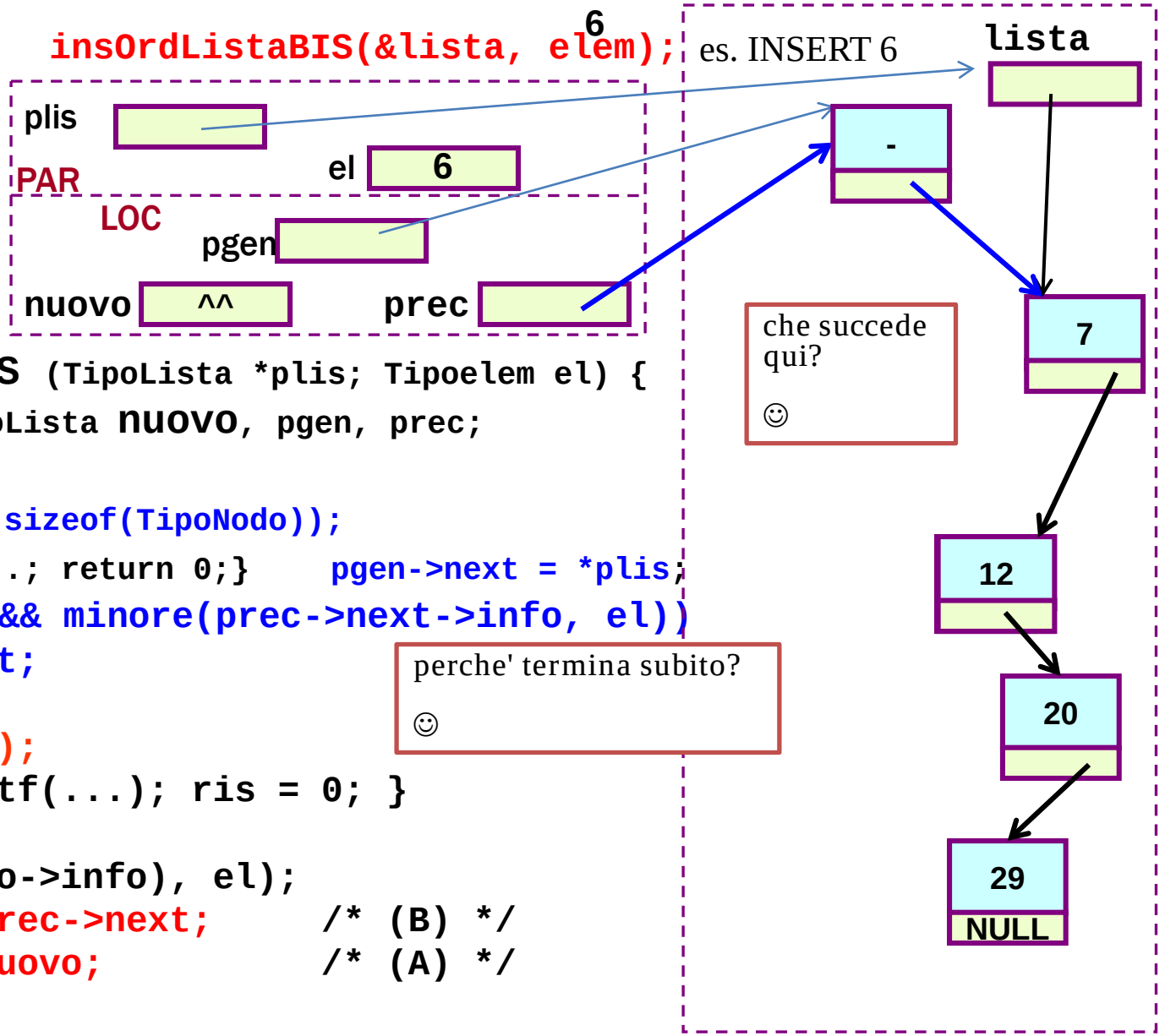

```
int insOrdListaBIS (TipoLista *plis; Tipoelem el) {
 int ris = 1; PuntNodoLista nuovo, pgen, prec;

 prec = pgen = malloc(sizeof(TipoNodo));
 if (!pgen) {printf ...; return 0;}    pgen->next = *plis;
 while (prec->next && minore(prec->next->info, el))
    prec = prec->next;

 nuovo = malloc(...);
  if (!nuovo) {printf(...); ris = 0; }
  else {
      assegna(&(nuovo->info), el);
     nuovo->next= prec->next;      /* (B) */
     prec->next = nuovo;           /* (A) */
}
 *plis = pgen->next;  free(pgen);  return ris; }
```

Inserimento ordinato in lista - uso di funzione di inserimento - alternativa 2/4

caso particolare: inserimento in testa

INSERT 6

il ciclo termina subito per fallimento della seconda condizione



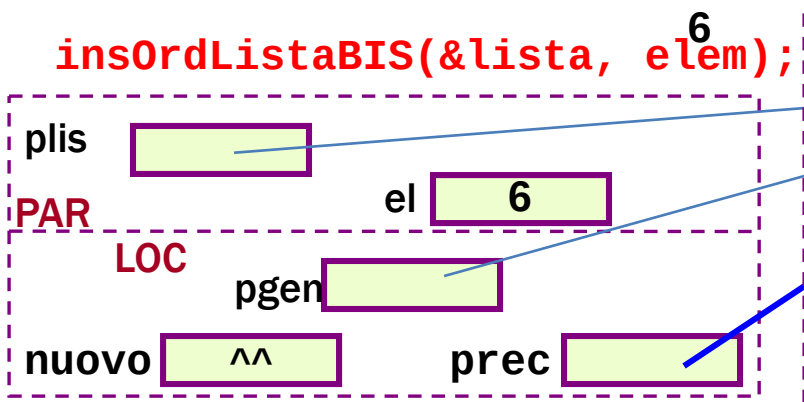

```
int insOrdListaBIS (TipoLista *plis; Tipoelem el) {
 int ris = 1; PuntNodoLista nuovo, pgen, prec;

 prec = pgen = malloc(sizeof(TipoNodo));
 if (!pgen) {printf ...; return 0;}    pgen->next = *plis;
 while (prec->next && minore(prec->next->info, el))
   prec = prec->next;

 nuovo = malloc(...);
  if (!nuovo) {printf(...); ris = 0; }
  else {
     assegna(&(nuovo->info), el);
     nuovo->next= prec->next;      /* (B) */
     prec->next = nuovo;           /* (A) */
}
 *plis = pgen->next;   free(pgen);  return ris; }
```

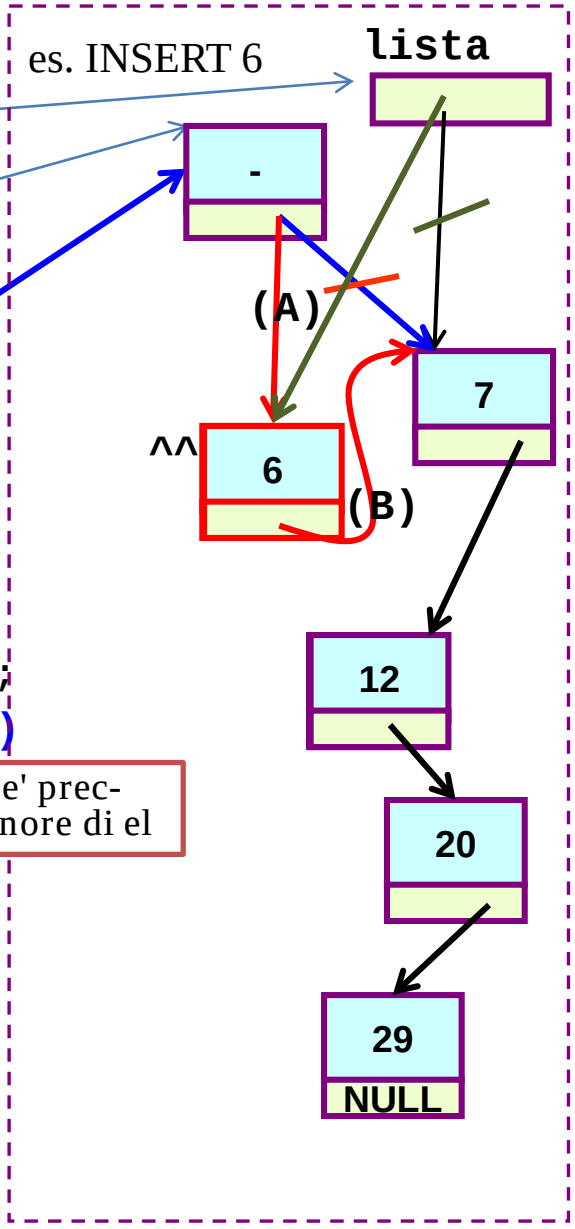
Inserimento ordinato in lista - uso di funzione di inserimento - alternativa 2/4

caso particolare: inserimento in testa

INSERT 6

il ciclo termina subito per fallimento della seconda condizione






```
int insOrdListaBIS (TipoLista *plis; Tipoelem el) {
 int ris = 1; PuntNodoLista nuovo, pgen, prec;

 prec = pgen = malloc(sizeof(TipoNodo));
 if (!pgen) {printf ...; return 0;}     pgen->next = *plis;
 while (prec->next && minore(prec->next->info, el))
   prec = prec->next;

 nuovo = malloc(...);
  if (!nuovo) {printf(...); ris = 0; }
  else {
     assegna(&(nuovo->info), el);
     nuovo->next= prec->next;        /* (B) */
     prec->next = nuovo;             /* (A) */
}
 *plis = pgen->next;   free(pgen);  return ris; }
```

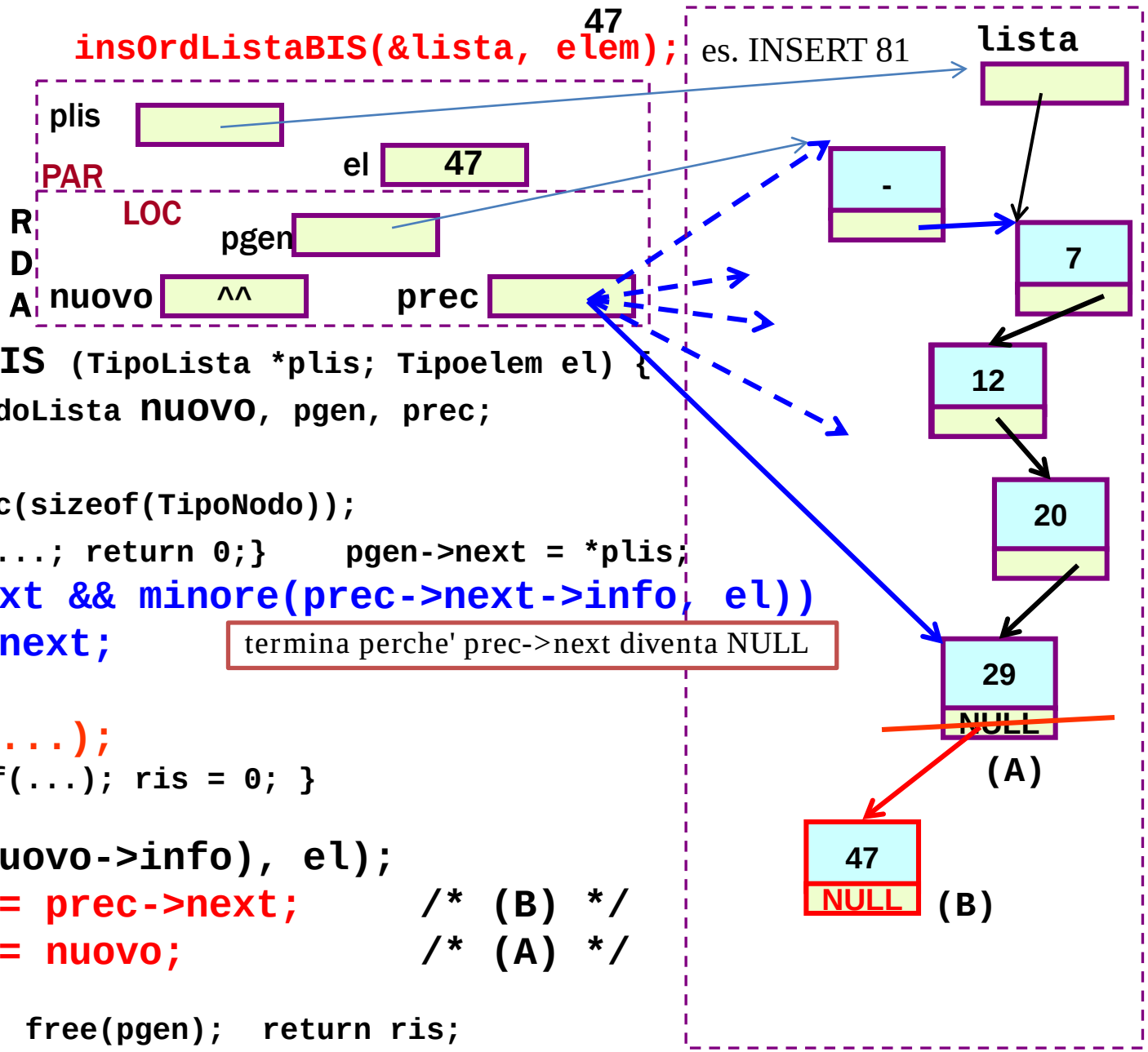
termina subito ... perche' prec->next->info non e` minore di el

Inserimento ordinato in lista - uso di funzione di inserimento - alternativa 3/4

caso particolare: inserimento in coda

INSERT 47

il ciclo termina dopo che la lista e` stata tutta scandita (cioe` con prec->next==NULL)




```
int insOrdListaBIS (TipoLista *plis; Tipoelem el) {
 int ris = 1; PuntNodoLista nuovo, pgen, prec;

 prec = pgen = malloc(sizeof(TipoNodo));
 if (!pgen) {printf ...; return 0;}    pgen->next = *plis;
 while (prec->next && minore(prec->next->info, el))
   prec = prec->next;

 nuovo = malloc(...);
  if (!nuovo) {printf(...); ris = 0; }
  else {
      assegna(&(nuovo->info), el);
      nuovo->next= prec->next;      /* (B) */
      prec->next = nuovo;           /* (A) */
}
 *plis = pgen->next;  free(pgen);  return ris;
```

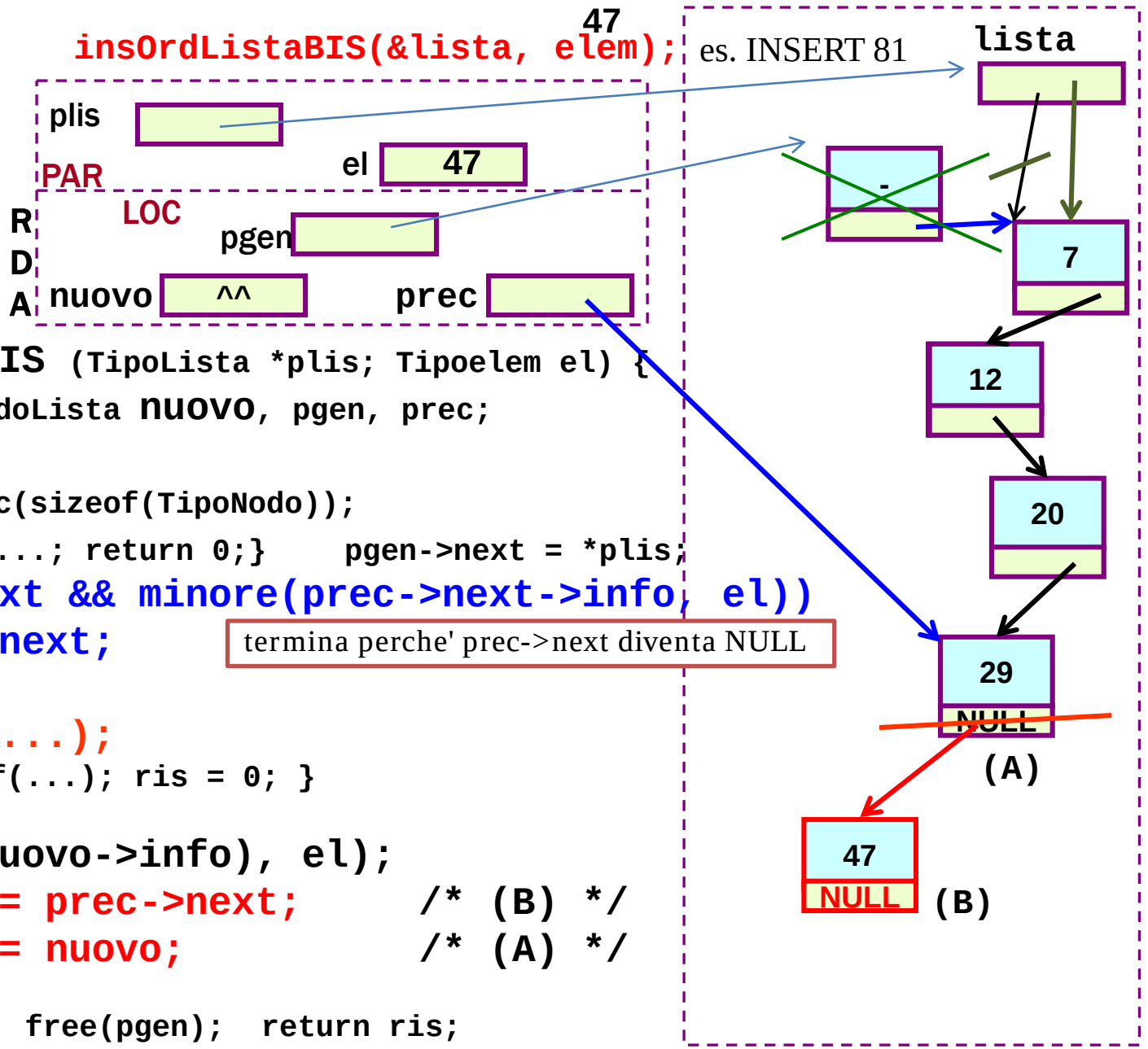
termina perche' prec->next diventa NULL

Inserimento ordinato in lista - uso di funzione di inserimento - alternativa 3/4

caso particolare: inserimento in coda

INSERT 47

il ciclo termina dopo che la lista e` stata tutta scandita (cioe` con prec->next==NULL)




```
int insOrdListaBIS (TipoLista *plis; Tipoelem el) {
 int ris = 1; PuntNodoLista nuovo, pgen, prec;

 prec = pgen = malloc(sizeof(TipoNodo));
 if (!pgen) {printf ...; return 0;}    pgen->next = *plis;
 while (prec->next && minore(prec->next->info, el))
   prec = prec->next;

 nuovo = malloc(...);
  if (!nuovo) {printf(...); ris = 0; }
  else {
      assegna(&(nuovo->info), el);
      nuovo->next= prec->next;      /* (B) */
      prec->next = nuovo;           /* (A) */
}
 *plis = pgen->next;  free(pgen);  return ris;
```

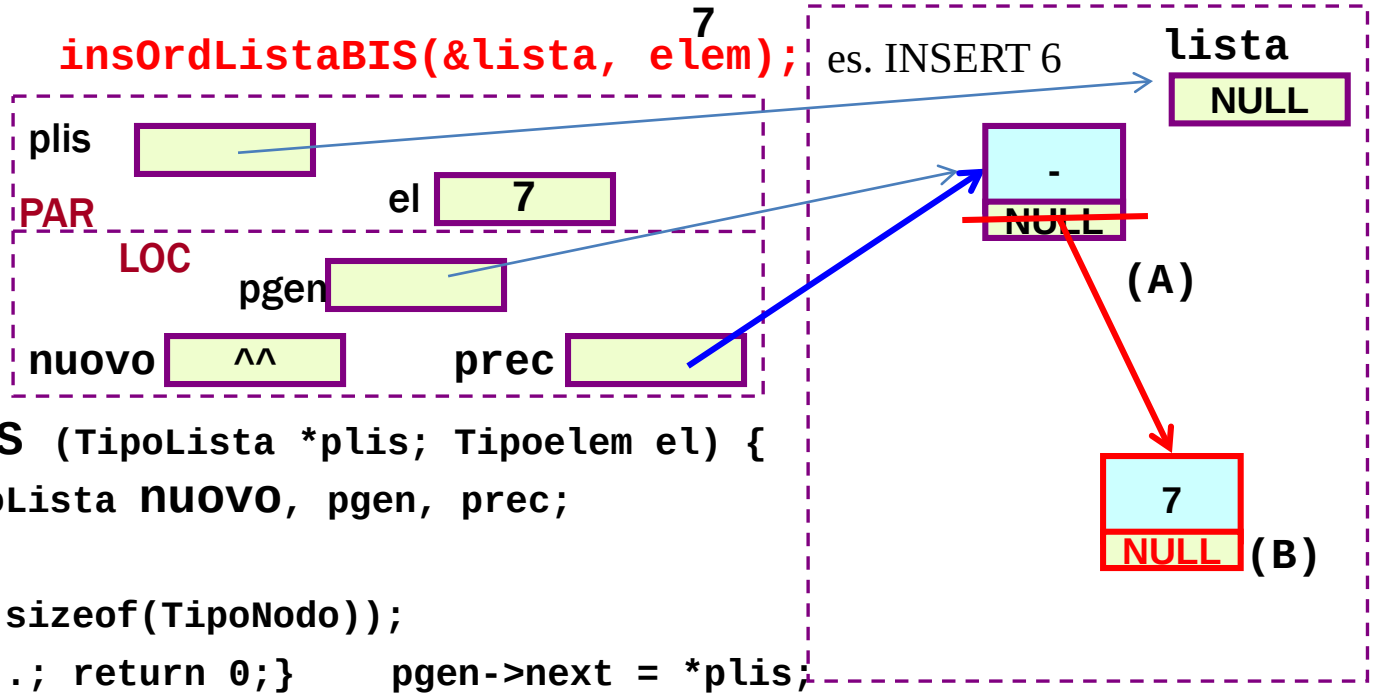
termina perche' prec->next diventa NULL

Inserimento ordinato in lista - uso di funzione di inserimento - alternativa 4/4

caso particolare: inserimento in lista vuota

INSERT 7

il ciclo termina subito. per il fallimento della prima condizione (prec->next!=NULL)




```
int insOrdListaBIS (TipoLista *plis; Tipoelem el) {
 int ris = 1; PuntNodoLista nuovo, pgen, prec;

 prec = pgen = malloc(sizeof(TipoNodo));
 if (!pgen) {printf ...; return 0;}    pgen->next = *plis;
 while (prec->next && minore(prec->next->info, el))
   prec = prec->next;

 nuovo = malloc(...);
  if (!nuovo) {printf(...); ris = 0; }
  else {
      assegna(&(nuovo->info), el);  /* il 7 */
      nuovo->next= prec->next;      /* (B)  */
      prec->next = nuovo;           /* (A)  */
}
 *plis = pgen->next;   free(pgen);  return ris; }
```

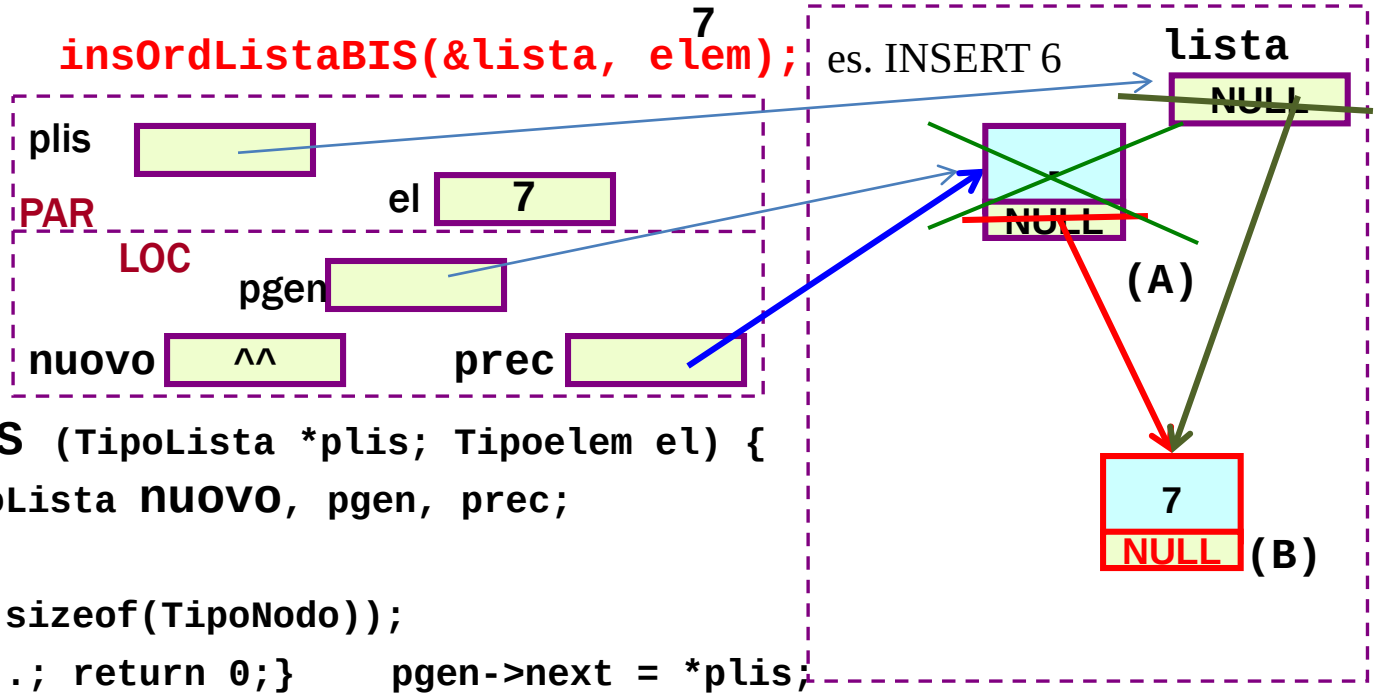
termina subito ... perche' prec->next e` NULL

Inserimento ordinato in lista - uso di funzione di inserimento - alternativa 4/4

caso particolare: inserimento in lista vuota

INSERT 7

il ciclo termina subito. per il fallimento della prima condizione (prec->next!=NULL)



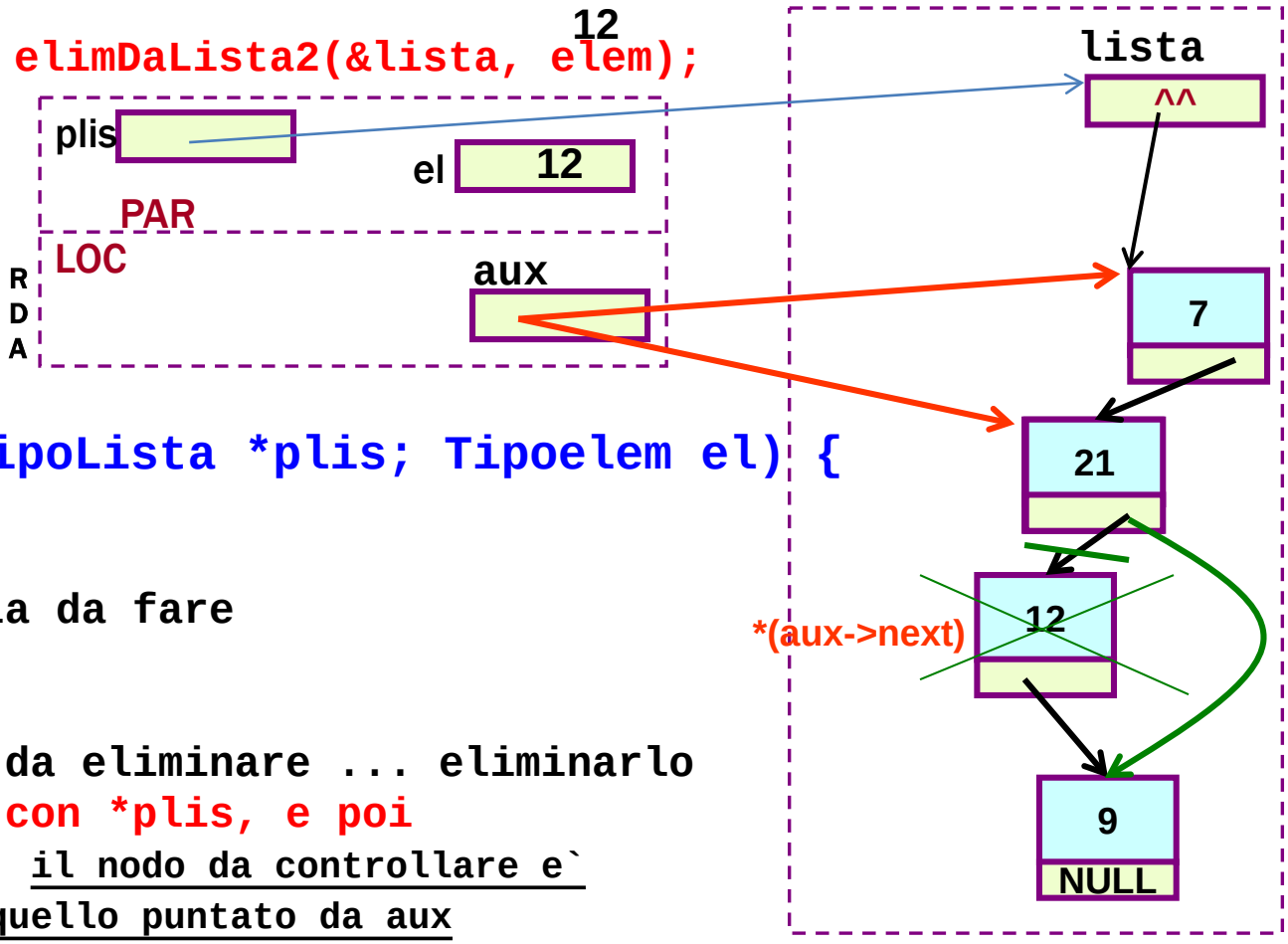

```
int insOrdListaBIS (TipoLista *plis; Tipoelem el) {
 int ris = 1; PuntNodoLista nuovo, pgen, prec;

 prec = pgen = malloc(sizeof(TipoNodo));
 if (!pgen) {printf ...; return 0;}    pgen->next = *plis;
 while (prec->next && minore(prec->next->info, el))
   prec = prec->next;

 nuovo = malloc(...);
  if (!nuovo) {printf(...); ris = 0; }
  else {
      assegna(&(nuovo->info), el);  /* il 7 */
      nuovo->next= prec->next;      /* (B)  */
      prec->next = nuovo;           /* (A)  */
}
 *plis = pgen->next;   free(pgen);  return ris; }
```

termina subito ... perche' prec->next e` NULL

Eliminazione di un elemento da una lista: funzione senza uso di RG 1/3

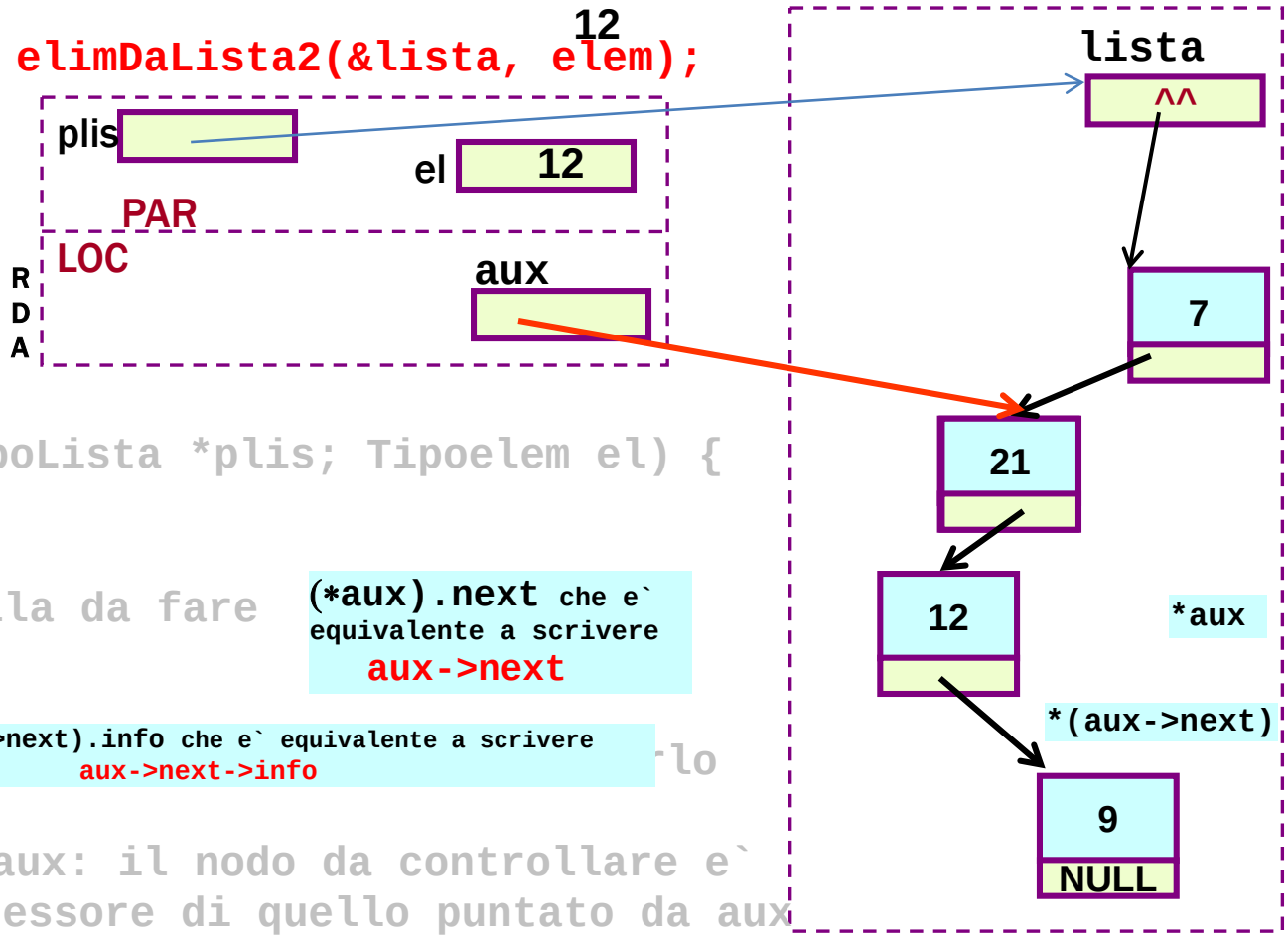
prima abbiamo usato il RG per uniformare casi in cui il codice da eseguire sarebbe stato diverso (caso generale e caso lista vuota)

se vogliamo fare a meno del RG, dobbiamo ripristinare la considerazione dei due casi distinti.




```
void elimDaLista2 (TipoLista *plis; Tipoelem el) {
 TipoNodo *aux, *aux2;

 - se lista vuota: nulla da fare

 - altrimenti
   se il primo nodo e` da eliminare ... eliminarlo
 -    senno`  init aux con *plis, e poi
        scansione con aux: il nodo da controllare e`
        il successore di quello puntato da aux
                                  --- cioe` *(aux->next)
        while( *(aux->next) ESISTE    e    aux->next->info != el)
           fai avanzare aux

    dopo il ciclo  - se aux->next->info==el  → ELIMINARE *(aux->next)
                   - se aux->next==NULL      → "nodo non trovato"
```

Eliminazione di un elemento da una lista: funzione senza uso di RG 1/3 – pit stop




```
void elimDaLista2 (TipoLista *plis; Tipoelem el) {
                  x, *aux2;
  - se lista vuota: nulla da fare

  - altrimenti
    se il primo nod                                    rlo
      senno`
        scansione con aux: il nodo da controllare e`
              il successore di quello puntato da aux
                        --- cioe` *(aux->next)
        while( *(aux->next) ESISTE    e    aux->next->info <> el)
          avanza aux

                                  x->next==NULL      → "nodo non trovato"
                          - aux->next->info==el →  ELIMINARE *(aux-
    >next)
```

(*aux).info che e` equivalente a scrivere **aux->info**

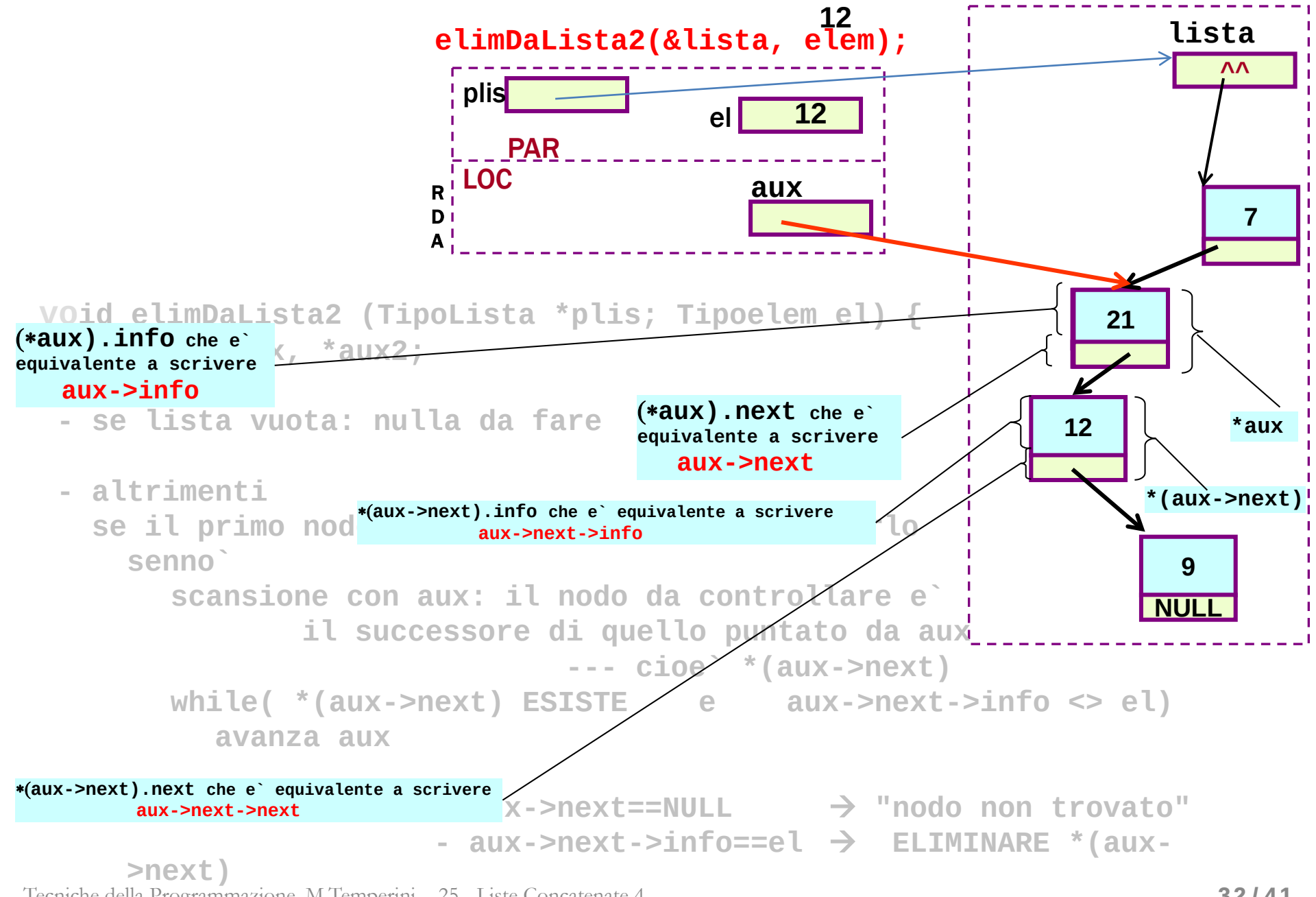
(*aux).next che e` equivalente a scrivere **aux->next**

***(aux->next).info** che e` equivalente a scrivere **aux->next->info**

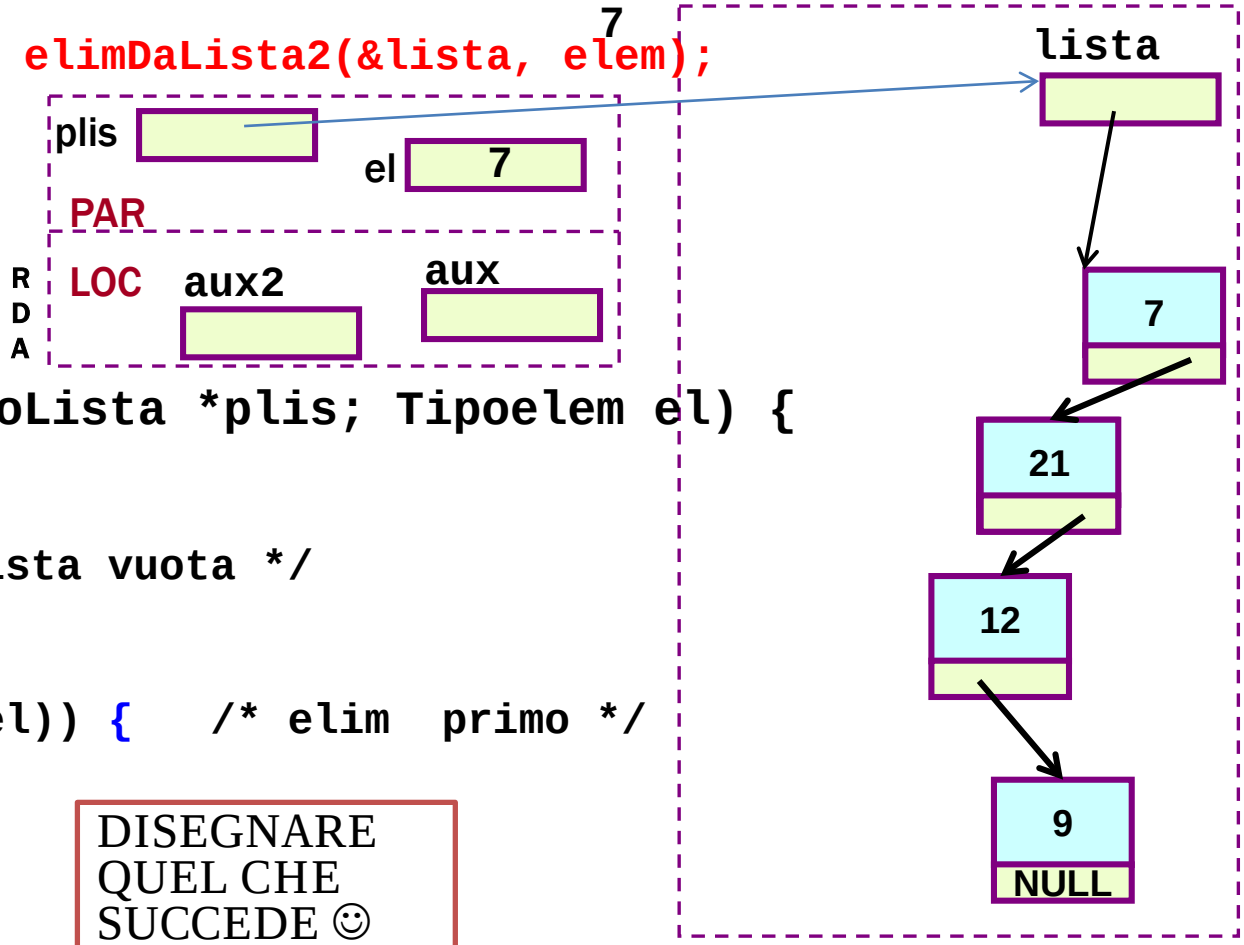
***(aux->next).next** che e` equivalente a scrivere **aux->next->next**

☺ a cosa corrispondono in figura le parti celesti ?

Eliminazione di un elemento da una lista: funzione senza uso di RG 1/3 – pit stop



Eliminazione di un elemento da una lista: funzione senza uso di RG 2/3

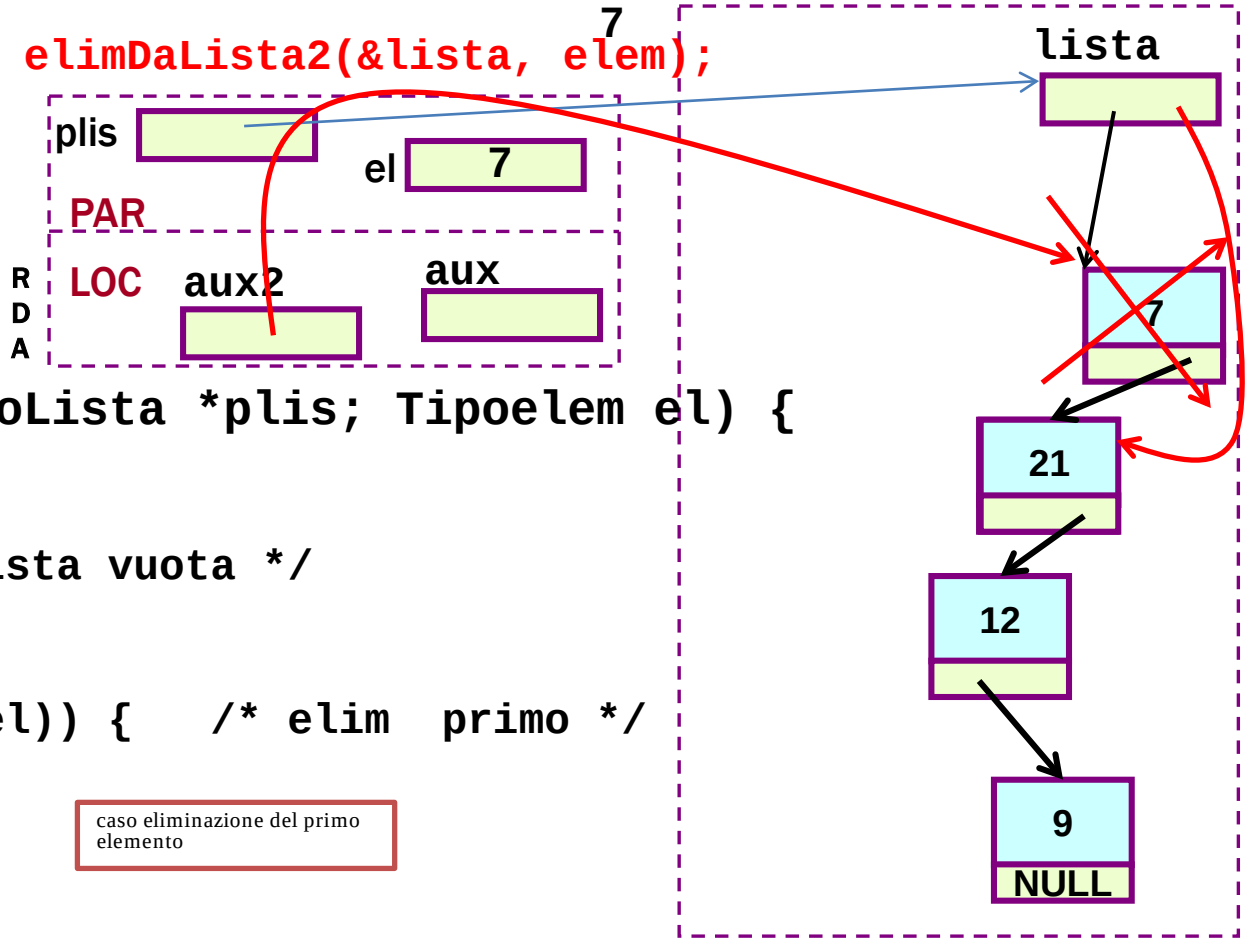



```
void elimDaLista2 (TipoLista *plis; Tipoelem el) {
 TipoNodo *aux, *aux2;

 if (*plis==NULL)   /* lista vuota */
   return;

if (uguali(*plis->info, el)) {   /* elim  primo */
   aux2= *plis;
   *plis = aux2->next;
   free(aux2);
   return;
}
```

DISEGNARE QUEL CHE SUCCEDE ☺

Eliminazione di un elemento da una lista: funzione senza uso di RG 2/3

abbiamo usato il RG per uniformare casi in cui il codice da eseguire sarebbe stato diverso (caso generale e caso lista vuota).
Se vogliamo fare a meno del RG, dobbiamo ripristinare la considerazione dei due casi distinti.

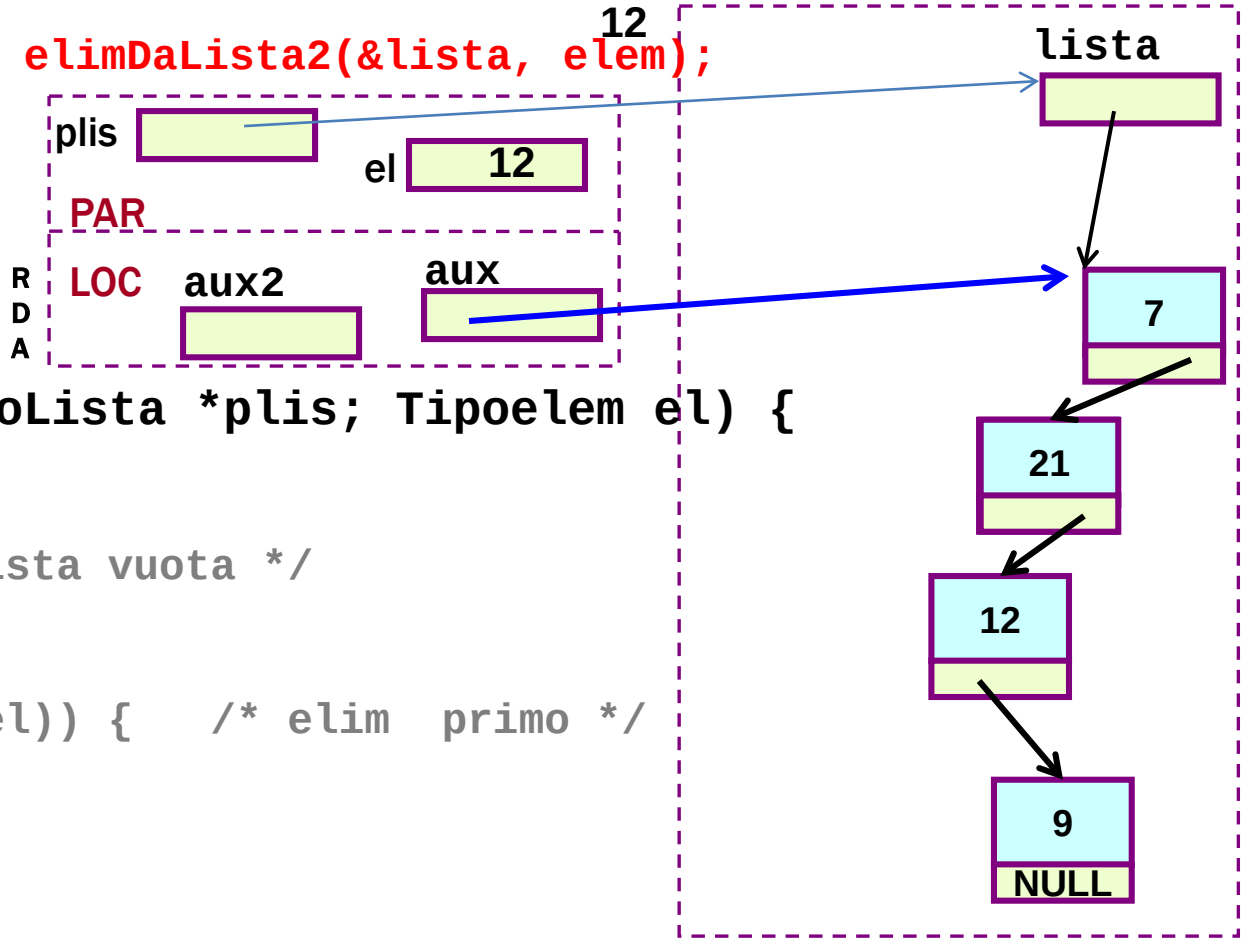



```
void elimDaLista2 (TipoLista *plis; Tipoelem el) {
 TipoNodo *aux, *aux2;

 if (*plis==NULL)   /* lista vuota */
   return;

if (uguali(*plis->info, el)) {   /* elim  primo */
   aux2= *plis;
   *plis = aux2->next;
   free(aux2);
   return;
}
```

caso eliminazione del primo elemento

Eliminazione di un elemento da una lista: funzione senza uso di RG 2/3

abbiamo usato il RG per uniformare casi in cui il codice da eseguire sarebbe stato diverso (caso generale e caso lista vuota).
Se vogliamo fare a meno del RG, dobbiamo ripristinare la considerazione dei due casi distinti.

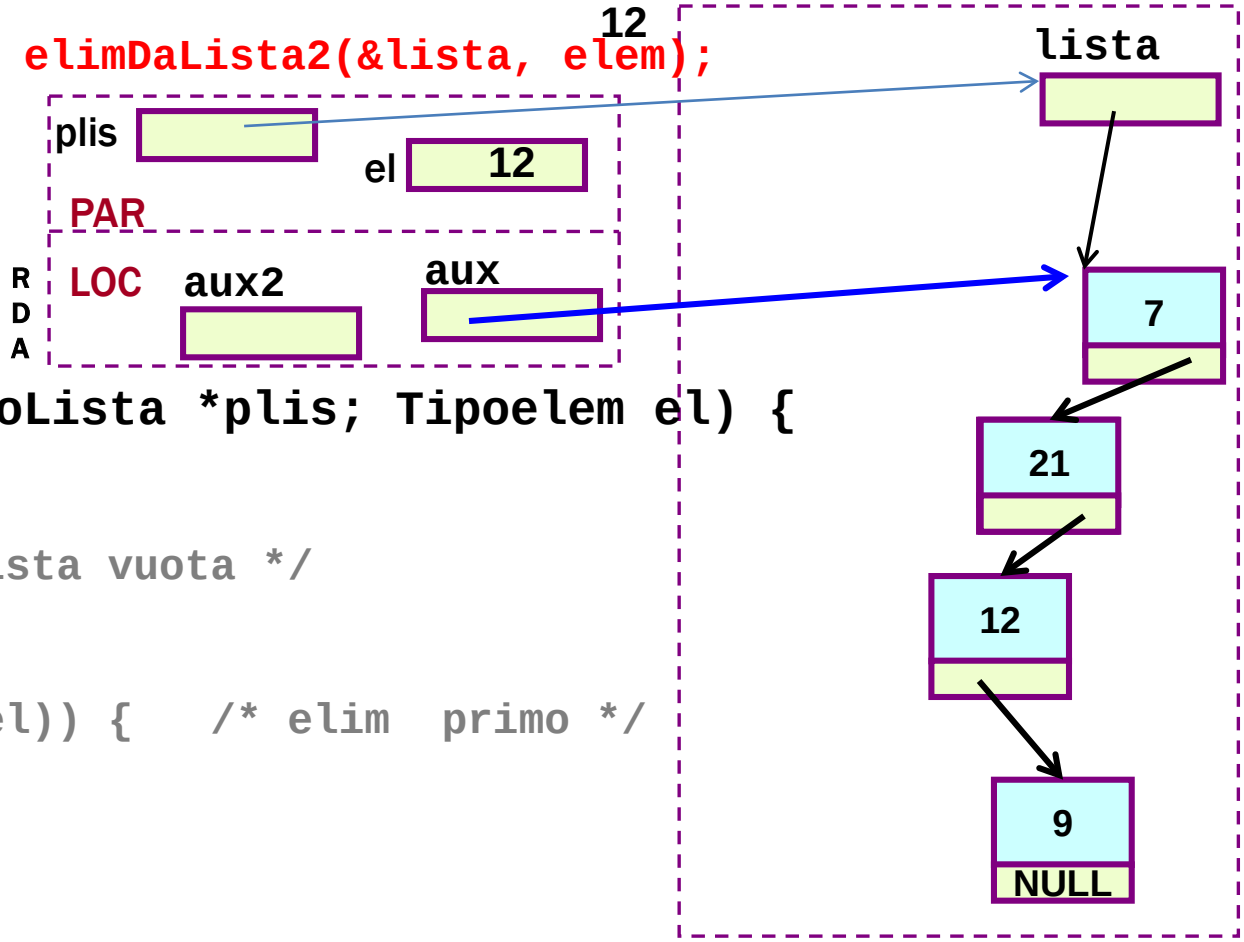



```
void elimDaLista2 (TipoLista *plis; Tipoelem el) {
 TipoNodo *aux, *aux2;

 if (*plis==NULL)   /* lista vuota */
   return;

if (uguali(*plis->info, el)) {   /* elim  primo */
   aux2= *plis;
   *plis = aux2->next;
   free(aux2);
   return;
}

aux = *plis;   /* NB il primo e` stato gia` controllato */

...
```

while(*(aux->next) ESISTE
e aux->next->info <> el)
avanza aux

Eliminazione di un elemento da una lista: funzione senza uso di RG 2/3

abbiamo usato il RG per uniformare casi in cui il codice da eseguire sarebbe stato diverso (caso generale e caso lista vuota).
Se vogliamo fare a meno del RG, dobbiamo ripristinare la considerazione dei due casi distinti.

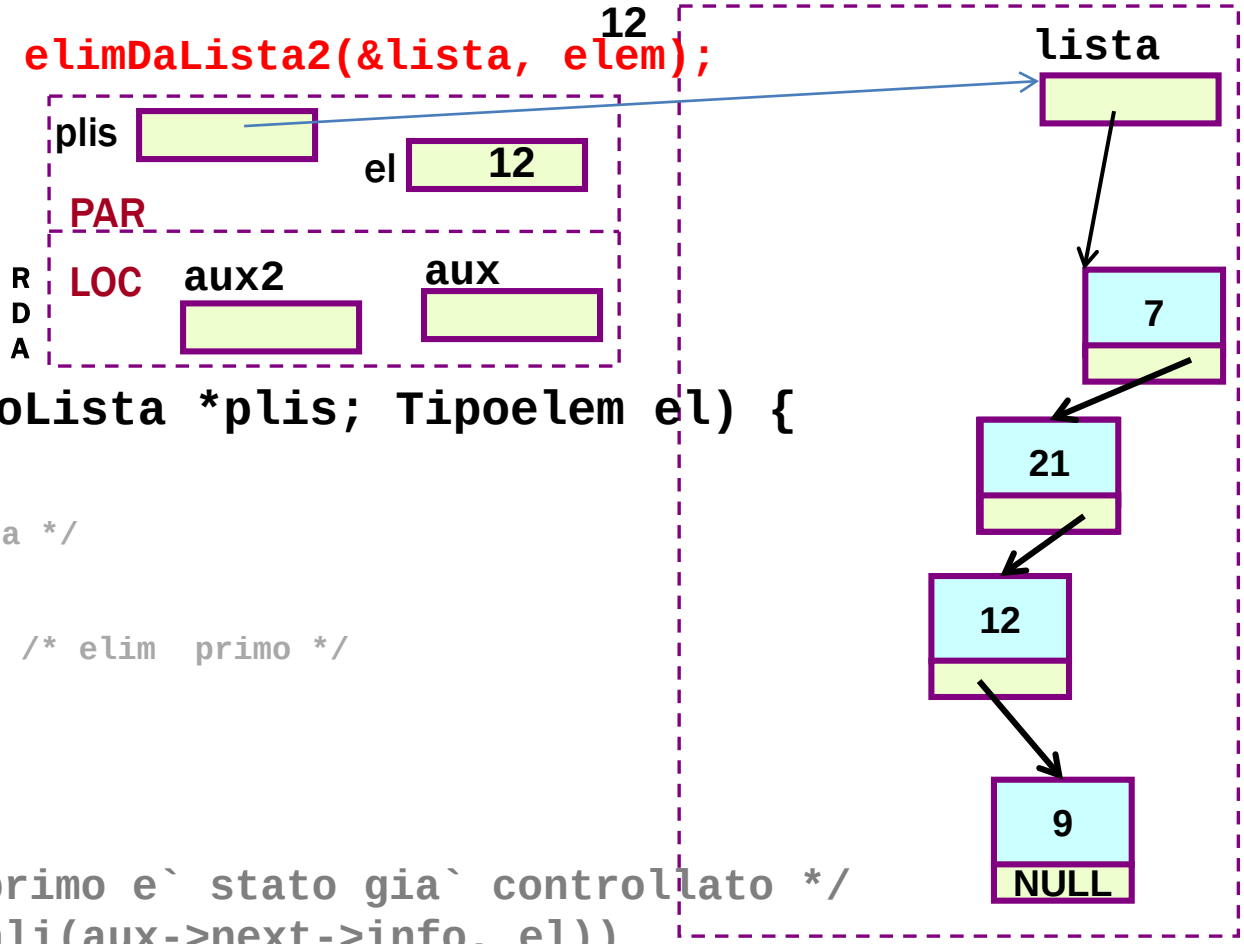



```
void elimDaLista2 (TipoLista *plis; Tipoelem el) {
 TipoNodo *aux, *aux2;

 if (*plis==NULL)   /* lista vuota */
   return;

if (uguali(*plis->info, el)) {   /* elim  primo */
   aux2= *plis;
   *plis = aux2->next;
   free(aux2);
   return;
}

aux = *plis;   /* NB il primo e` stato gia` controllato */

while( aux->next && !uguali(aux->next->info, el))
...
```

while(*(aux->next) ESISTE
e aux->next->info <> el)
avanza aux

Eliminazione di un elemento da una lista: funzione senza uso di RG 3/3

abbiamo usato il RG per uniformare casi in cui il codice da eseguire sarebbe stato diverso (caso generale e caso lista vuota).
Se vogliamo fare a meno del RG, dobbiamo ripristinare la considerazione dei due casi distinti.



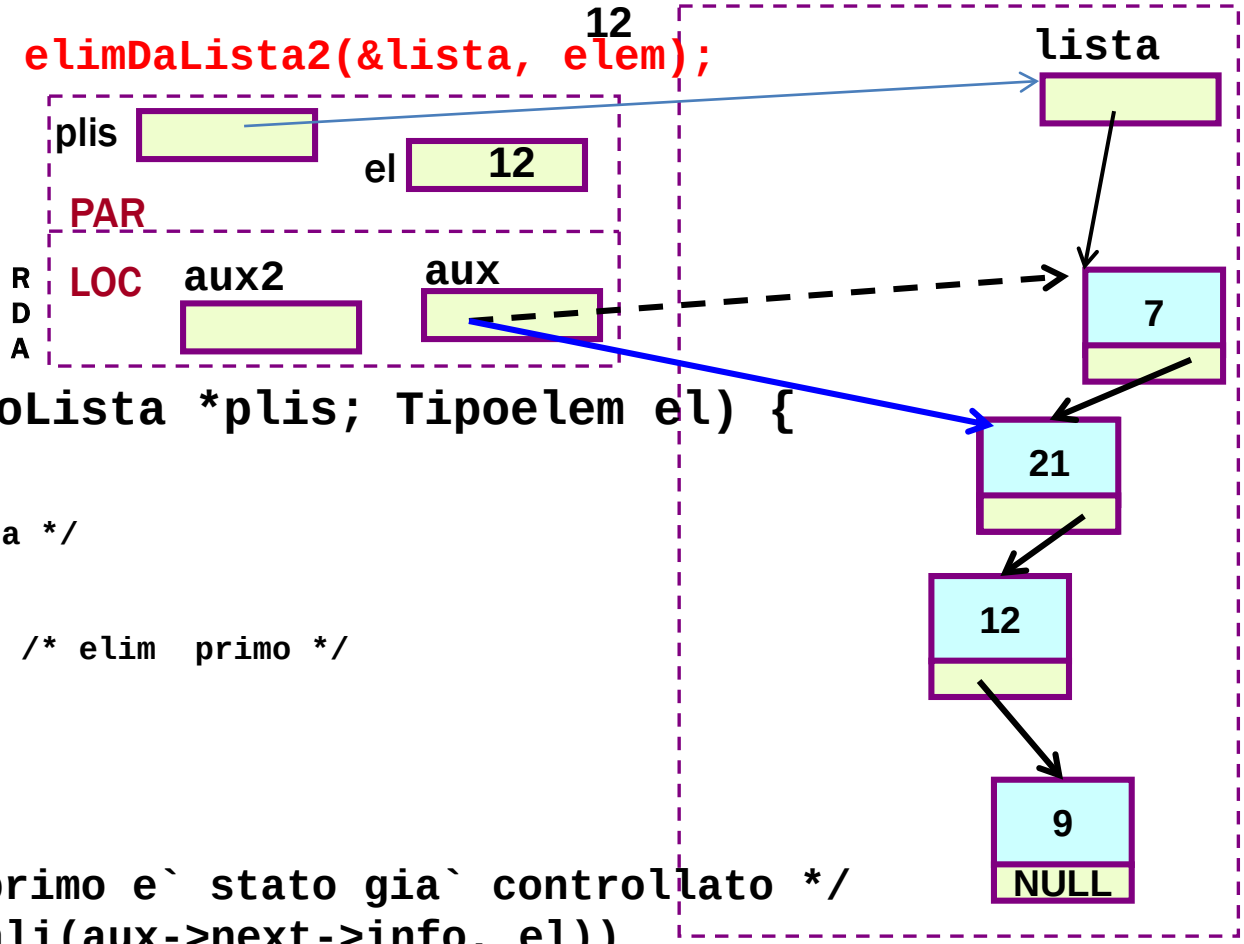

```
void elimDaLista2 (TipoLista *plis; Tipoelem el) {
 TipoNodo *aux, *aux2;

 if (*plis==NULL)   /* lista vuota */
   return;

 if (uguali(*plis->info, el)) {   /* elim  primo */
   aux2= *plis;
   *plis = aux2->next;
   free(aux2);
   return;
 }
aux = *plis;   /* NB il primo e` stato gia` controllato */
while( aux->next && !uguali(aux->next->info, el))
  aux = aux->next;
 /* ora o aux-next e` NULL o punta sul nodo da eliminare */
if (aux->next) { aux2=aux->next;
                 aux->next = aux->next->next;
                 free(aux2);
}      return; }
```

caso eliminazione di elemento interno alla lista

completare a mano il disegno, eseguendo il codice da aux=*plis in poi

Eliminazione di un elemento da una lista: funzione senza uso di RG 3/3

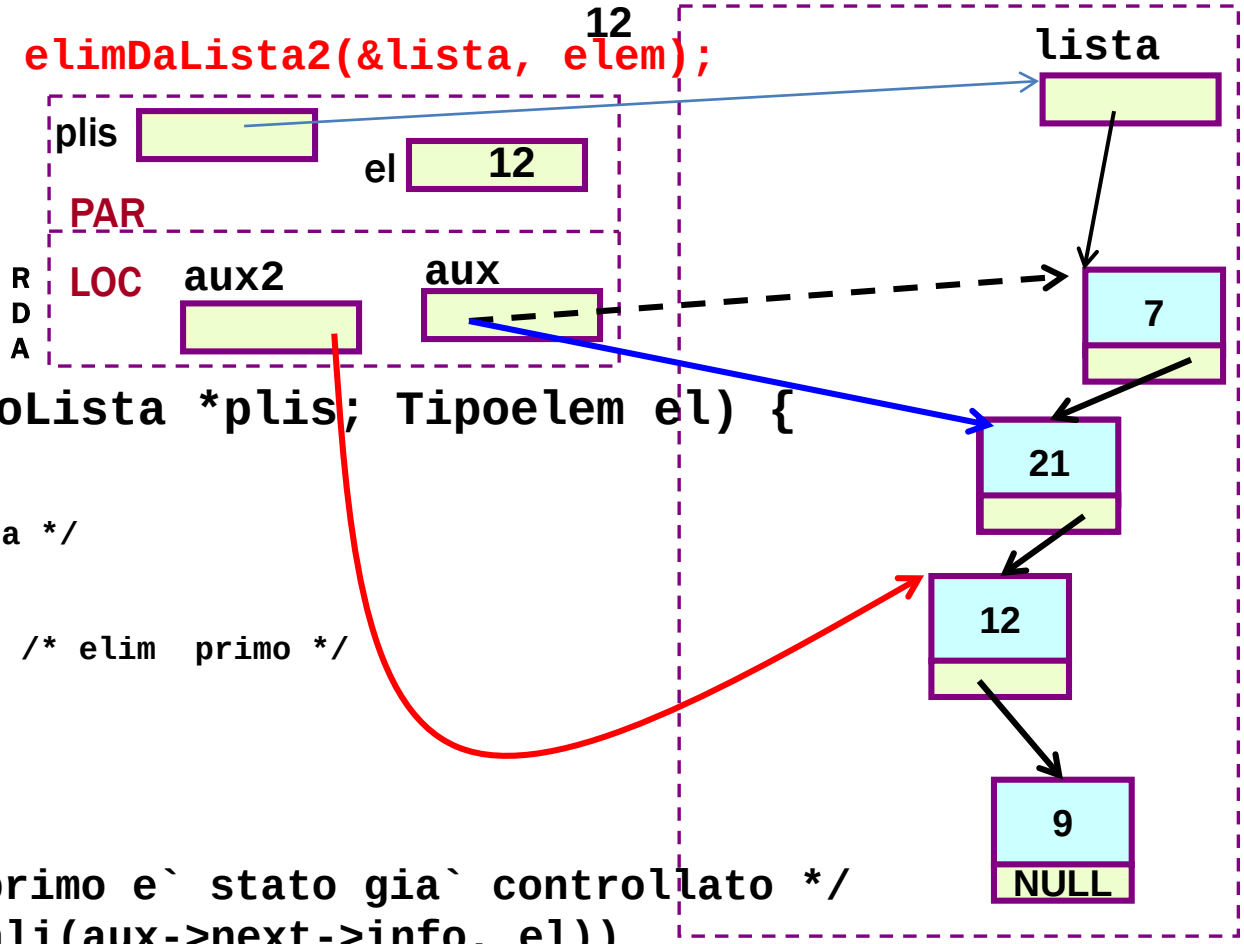
abbiamo usato il RG per uniformare casi in cui il codice da eseguire sarebbe stato diverso (caso generale e caso lista vuota).
Se vogliamo fare a meno del RG, dobbiamo ripristinare la considerazione dei due casi distinti.




```
void elimDaLista2 (TipoLista *plis; Tipoelem el) {
 TipoNodo *aux, *aux2;

 if (*plis==NULL)   /* lista vuota */
   return;

 if (uguali(*plis->info, el)) {   /* elim  primo */
   aux2= *plis;
   *plis = aux2->next;
   free(aux2);
   return;
 }
aux = *plis;   /* NB il primo e` stato gia` controllato */
while( aux->next && !uguali(aux->next->info, el))
  aux = aux->next;
 /* ora o aux-next e` NULL o punta sul nodo da eliminare */
if (aux->next) { aux2=aux->next;
                 aux->next = aux->next->next;
                 free(aux2);
}                                                  return;}
```

Eliminazione di un elemento da una lista: funzione senza uso di RG 3/3

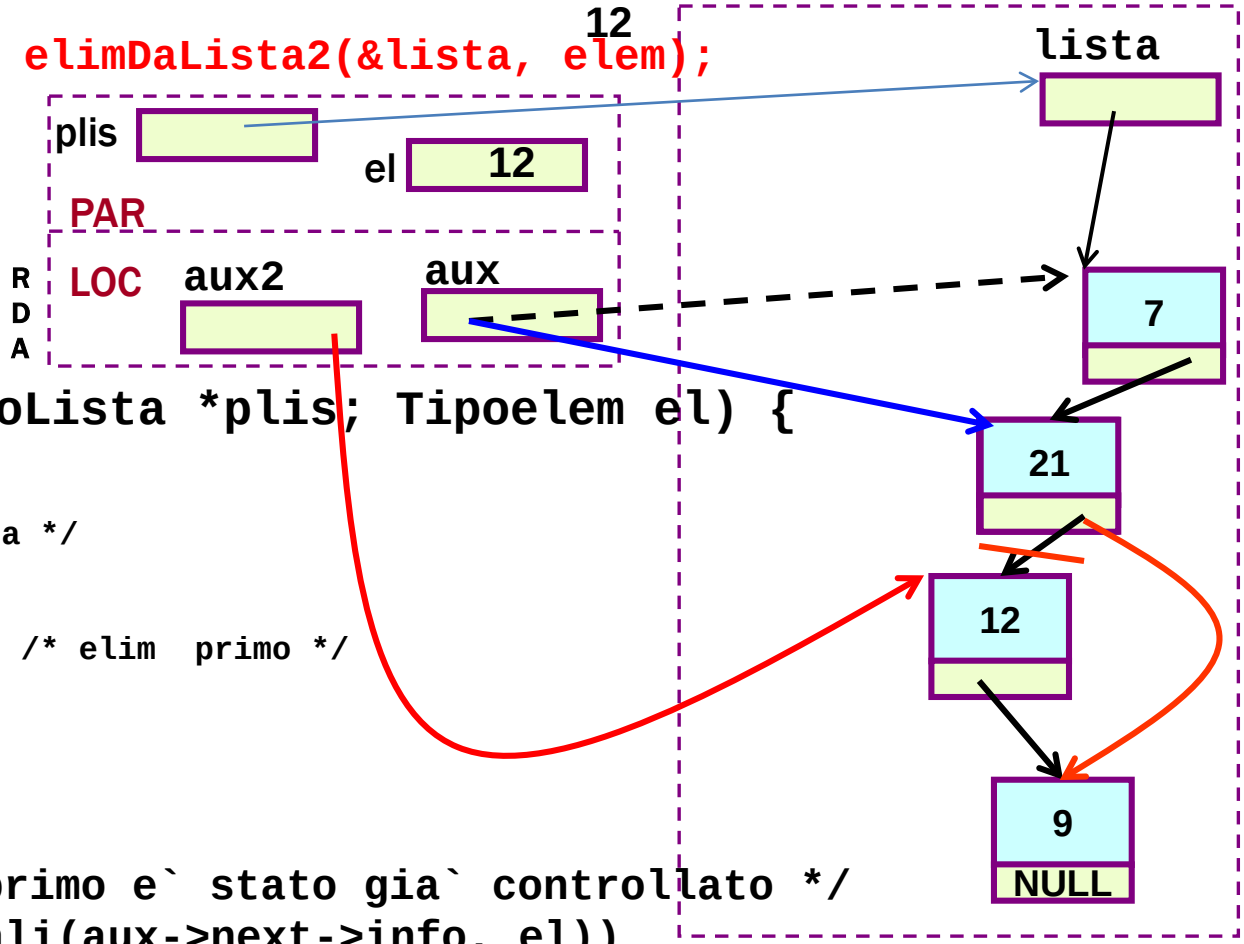
abbiamo usato il RG per uniformare casi in cui il codice da eseguire sarebbe stato diverso (caso generale e caso lista vuota).
Se vogliamo fare a meno del RG, dobbiamo ripristinare la considerazione dei due casi distinti.




```
void elimDaLista2 (TipoLista *plis; Tipoelem el) {
 TipoNodo *aux, *aux2;

 if (*plis==NULL)   /* lista vuota */
   return;

 if (uguali(*plis->info, el)) {   /* elim  primo */
   aux2= *plis;
   *plis = aux2->next;
   free(aux2);
   return;
 }
aux = *plis;   /* NB il primo e` stato gia` controllato */
while( aux->next && !uguali(aux->next->info, el))
  aux = aux->next;
 /* ora o aux-next e` NULL o punta sul nodo da eliminare */
if (aux->next) { aux2=aux->next;
                 aux->next = aux->next->next;
                 free(aux2);
}                                                  return;}
```

Eliminazione di un elemento da una lista: funzione senza uso di RG 3/3

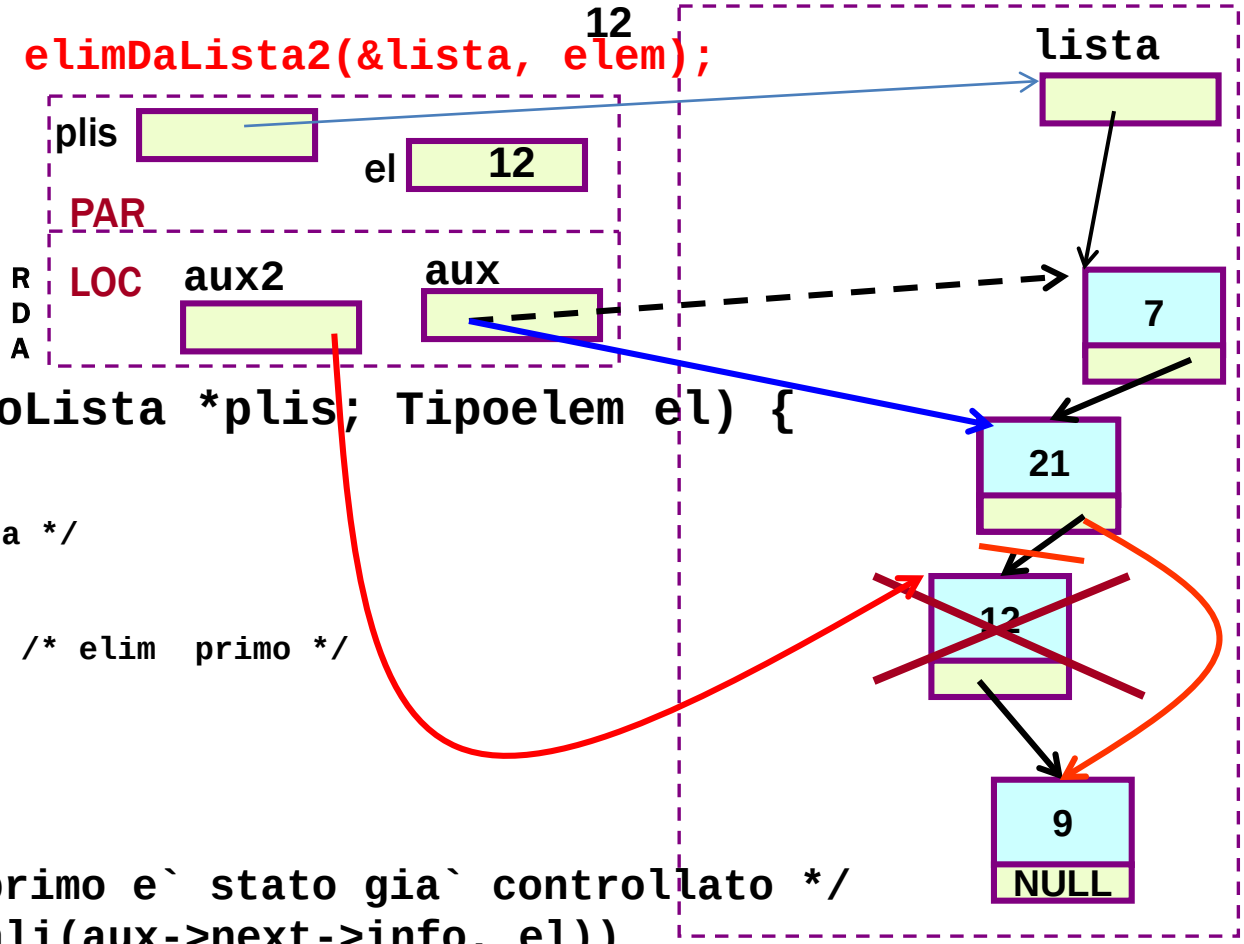
abbiamo usato il RG per uniformare casi in cui il codice da eseguire sarebbe stato diverso (caso generale e caso lista vuota).
Se vogliamo fare a meno del RG, dobbiamo ripristinare la considerazione dei due casi distinti.




```
void elimDaLista2 (TipoLista *plis; Tipoelem el) {
 TipoNodo *aux, *aux2;

 if (*plis==NULL)   /* lista vuota */
   return;

 if (uguali(*plis->info, el)) {   /* elim  primo */
   aux2= *plis;
   *plis = aux2->next;
   free(aux2);
   return;
 }
aux = *plis;   /* NB il primo e` stato gia` controllato */
while( aux->next && !uguali(aux->next->info, el))
  aux = aux->next;
 /* ora o aux-next e` NULL o punta sul nodo da eliminare */
if (aux->next) { aux2=aux->next;
                 aux->next = aux->next->next;
                 free(aux2);
}                                                  return;}
```

Eliminazione di un elemento da una lista: funzione senza uso di RG 3/3

abbiamo usato il RG per uniformare casi in cui il codice da eseguire sarebbe stato diverso (caso generale e caso lista vuota).
Se vogliamo fare a meno del RG, dobbiamo ripristinare la considerazione dei due casi distinti.




```
void elimDaLista2 (TipoLista *plis; Tipoelem el) {
 TipoNodo *aux, *aux2;

 if (*plis==NULL)   /* lista vuota */
   return;

 if (uguali(*plis->info, el)) {   /* elim  primo */
   aux2= *plis;
   *plis = aux2->next;
   free(aux2);
   return;
 }
aux = *plis;   /* NB il primo e` stato gia` controllato */
while( aux->next && !uguali(aux->next->info, el))
  aux = aux->next;
 /* ora o aux-next e` NULL o punta sul nodo da eliminare */
if (aux->next) { aux2=aux->next;
                 aux->next = aux->next->next;
                 free(aux2);
}                                              return;}
```