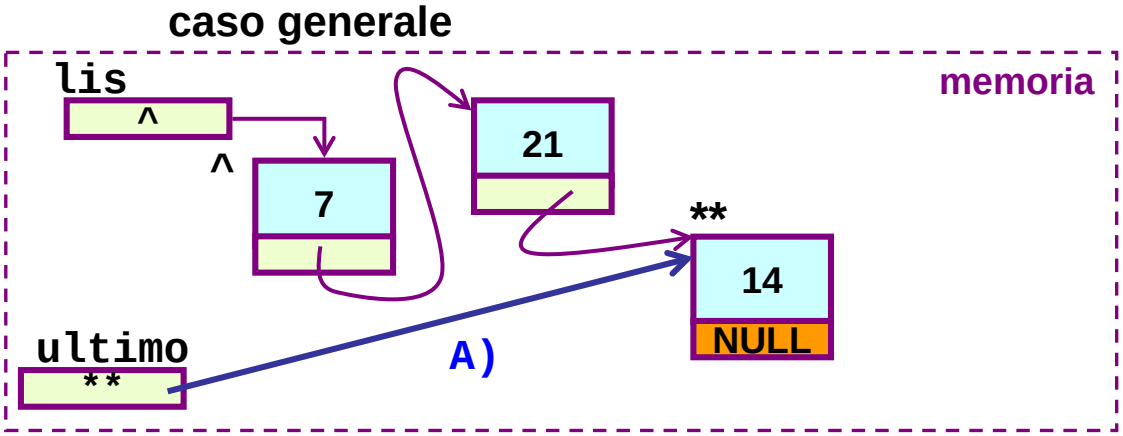
# Funzione di Inserimento in coda - 1/6 -



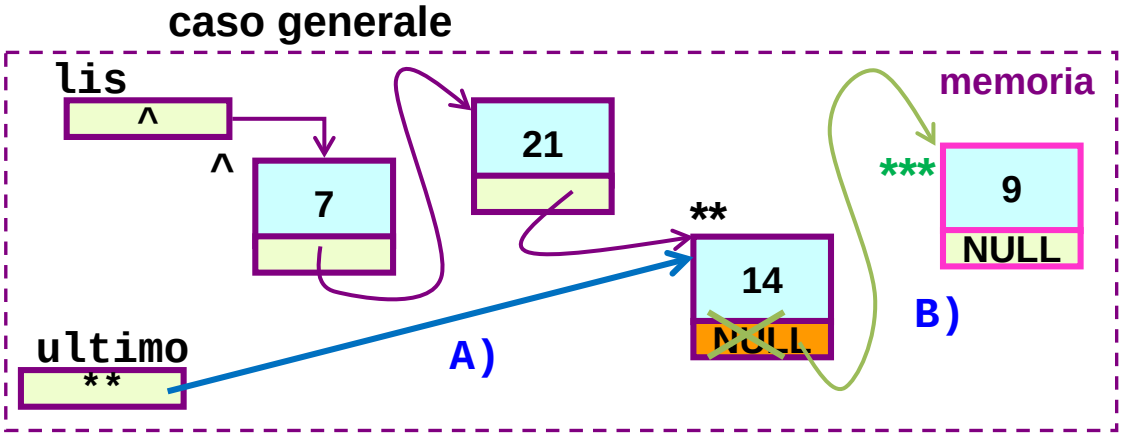

nel caso generico (lista non vuota) l'inserimento in coda non provoca una variazione del puntatore all'inizio della lista; MA NEL CASO DI INSERIMENTO IN LISTA VUOTA, SI`;
quindi la chiamata da effettuare e`

```
insInCoda(&lis, 9)
```

**Algoritmo**

```
A)  posizionare puntatore ultimo,
    sull'ultimo elemento della lista

B)  aggiungere nuovo elemento (con 9)
    dopo il nodo *ultimo
```

# Funzione di Inserimento in coda - 1/6 -



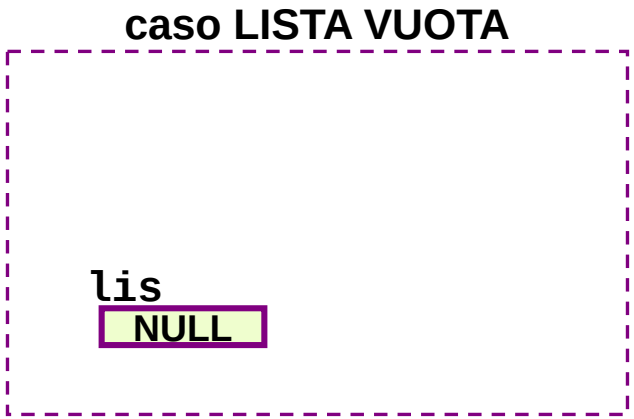

nel caso generico (lista non vuota) l'inserimento in coda non provoca una variazione del puntatore all'inizio della lista; MA NEL CASO DI INSERIMENTO IN LISTA VUOTA, Si`;
quindi la chiamata da effettuare e`

```
insInCoda(&lis, 9)
```

**Algoritmo**

```
A)  posizionare puntatore ultimo,
    sull'ultimo elemento della lista

B)  aggiungere nuovo elemento (con 9)
    dopo il nodo *ultimo
```

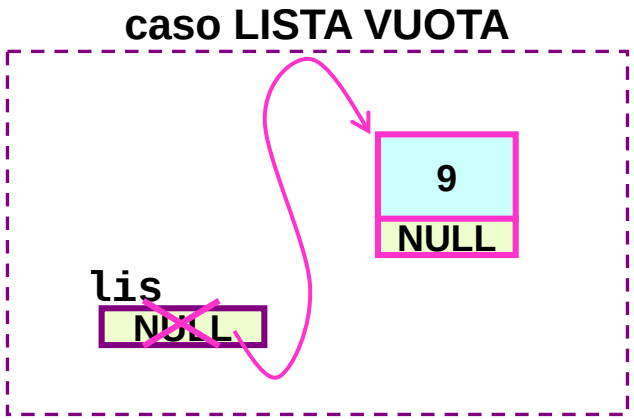
# Funzione di Inserimento in coda - 1/6 -




nel caso generico (lista non vuota) l'inserimento in coda non provoca una variazione del puntatore all'inizio della lista; MA NEL CASO DI INSERIMENTO IN LISTA VUOTA, Si`;
quindi la chiamata da effettuare e`

```
insInCoda(&lis, 9)
```

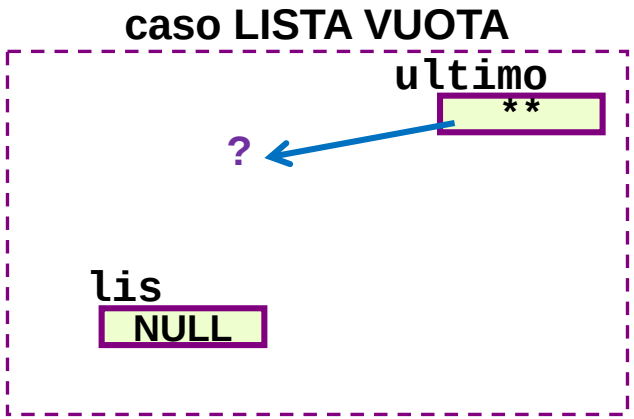
# Funzione di Inserimento in coda - 1/6 -




nel caso generico (lista non vuota) l'inserimento in coda non provoca una variazione del puntatore all'inizio della lista; MA NEL CASO DI INSERIMENTO IN LISTA VUOTA, Si`;
quindi la chiamata da effettuare e`

```
insInCoda(&lis, 9)
```

# Funzione di Inserimento in coda - 1/6 -



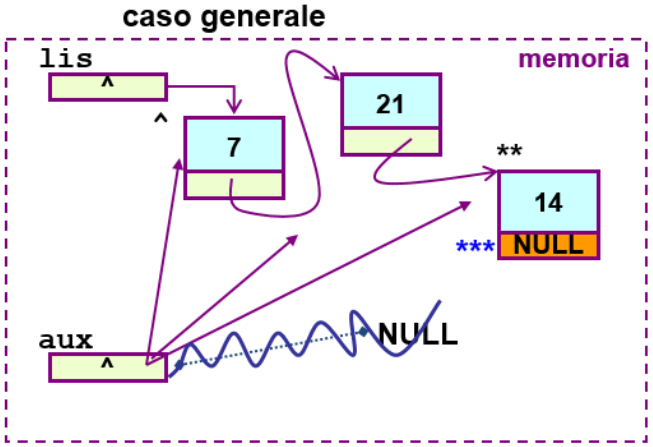

nel caso generico (lista non vuota) l'inserimento in coda non provoca una variazione del puntatore all'inizio della lista; MA NEL CASO DI INSERIMENTO IN LISTA VUOTA, Si`;
quindi la chiamata da effettuare e`

```
insInCoda(&lis, 9)
```

**Algoritmo**

```
A)  posizionare puntatore ultimo,
    sull'ultimo elemento della lista

B)  aggiungere nuovo elemento (con 9)
    dopo il nodo *ultimo
```

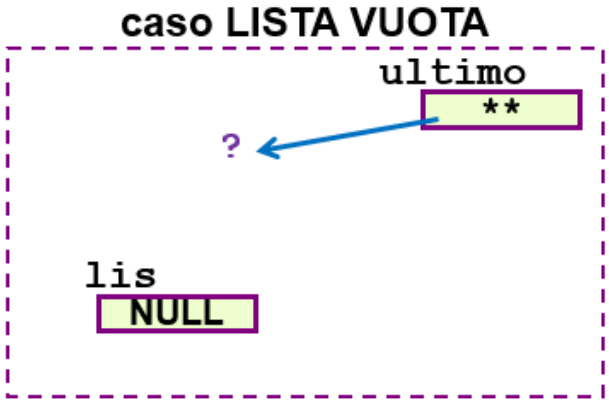
se volessimo usare i passi A e B non sapremmo dove far puntare ultimo all'inizio (nel caso LISTA VUOTA)




**posizionamento di ultimo (punto A)**

```
TipoNodo * aux;
aux=lis;
while (aux->next!=NULL)
    aux = aux->next;
/* ora aux punta sull'ultimo
     nodo */
```

**soluzione giusta - la condizione di avanzamento di aux non e` che aux sia non nullo, ma e` che aux punti ad un nodo che ha un successore; se il nodo cui punta aux non ha successore, allora lui e` l'ultimo (e aux punta all'ultimo)**

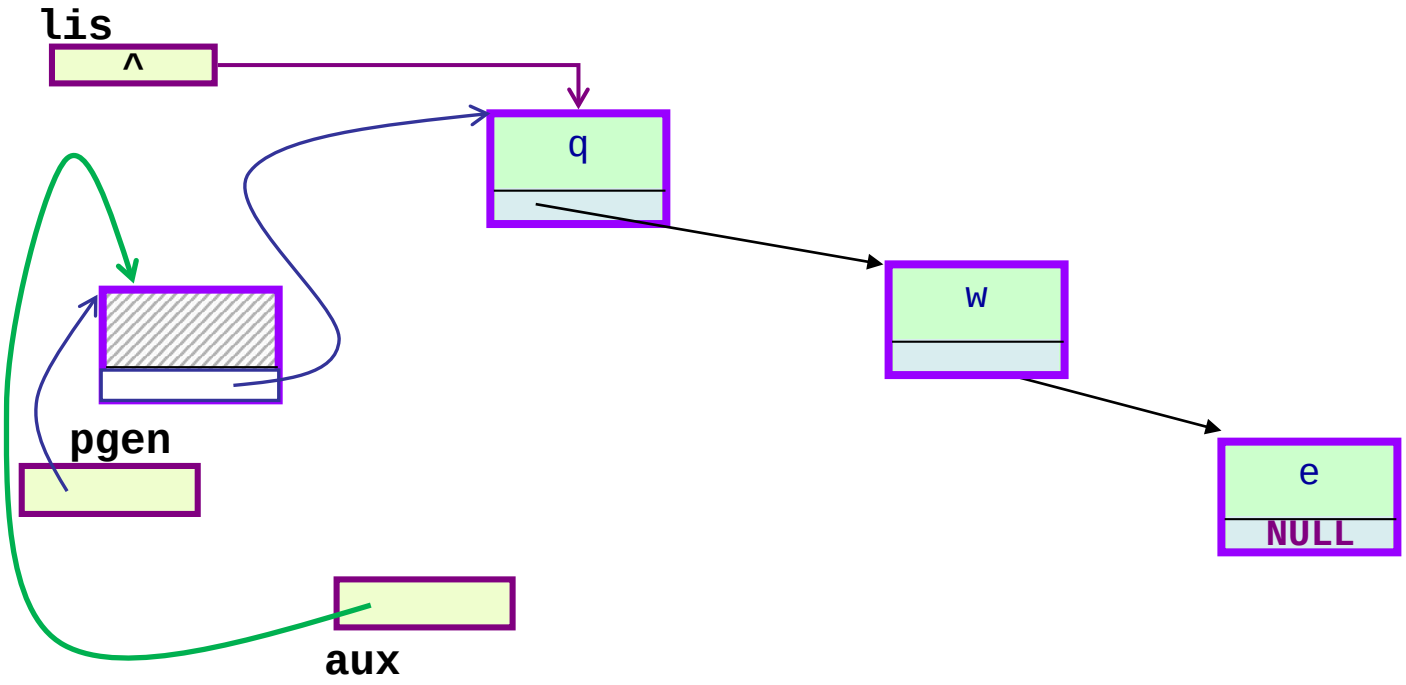



MA questo va bene solo se
la lista non e` vuota ... provare con
- lista di 6 elementi
- lista di un elemento
- lista vuota

se la lista e` vuota, dove punta aux all'inizio,
per permettere di valutare sensatamente **`aux->next`** ??

Allora usiamo sempre (caso generale e caso particolare) la tecnica del ...

# (applicazione tecnica del Record Generatore (RG) --- CASO GENERICO




```
TipoNodo * aux;
aux = pgen;
while (aux->next!=NULL)
    aux = aux->next;
/* all'inizio del ciclo aux punta su un nodo per il quale "->next"
     esiste … ed e` diverso da null. Ora possiamo procedere a muovere
     aux fino a farlo puntare sull'ultimo nodo e aggiungere il nuovo
     nodo in coda */
```

```
void insInCoda(TipoLista * plis, TipoElem el) {
```

**(Algoritmo che unifica i casi lista vuota e generico con la tecnica del RG)**

**0) pgen, ultimo, aux …**

**1) allocazione record generatore (RG) e suo posizionamento in testa alla lista;**

**2=A) inizializzazione aux**
**scansione per portare aux a puntare sull'ultimo**
**e poi assegnarvi ultimo**

**3=B) aggiunta dopo *ultimo (il nodo puntato da ultimo)**

```
2.1) ultimo->next = malloc(…)
2.2) ultimo = ultimo->next
2.3) ultimo->info = el
```

**4) chiusura lista - `ultimo->next = NULL`**

**5) sistemazione *plis** (side effect, assegnandogli `pgen->next`)
**6) eliminazione RG**

# And now … Lezione 24 – LISTE-3

**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

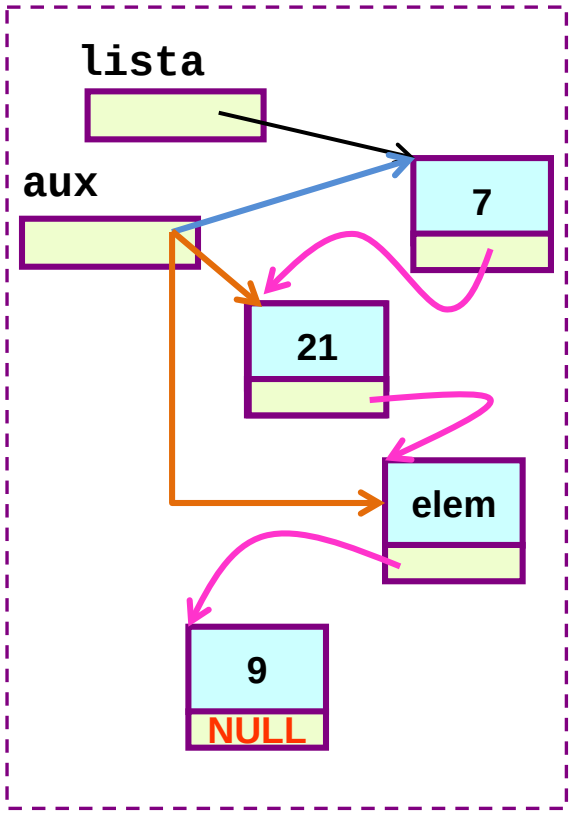
- eliminazione di un elemento da lista
  - versione alternativa: senza record generatore - elimDaLista2()

# Eliminazione di un elemento da una lista (nella funzione main())

```
...typedef int TipoElem;...

int main() {
 TipoLista lista;
 TipoElem elem;
...
  /* costruzione lista */
...
  leggiElem(&elem);  /* l'el. da eliminare */
...
... /* codice Δ che esegue l'eliminazione */
...
  stampaLista(lista);
...
...
return 0;
}
```

il programma, ad un certo punto gestisce l'eliminazione di un elemento dalla lista (elemento specificato in input)

# eliminazione di elem da lista: prima ipotesi




scansione con aux,
per cercare elem

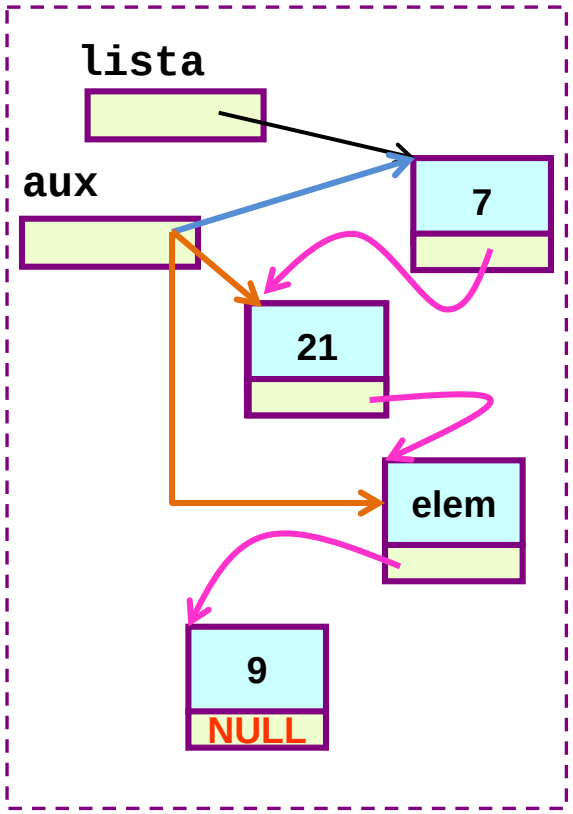
```
aux=lista;
while (aux && aux->info!=elem)
  aux = aux->next;
```

ora aux punta
sul nodo da eliminare

```
if (aux)
  free(aux);
```

☺ ??

# eliminazione di elem da lista: uhm ...




scansione con aux,
per cercare elem

```
aux=lista;
while (aux && aux->info!=elem)
   aux = aux->next;
```

ora aux punta
sul nodo da eliminare

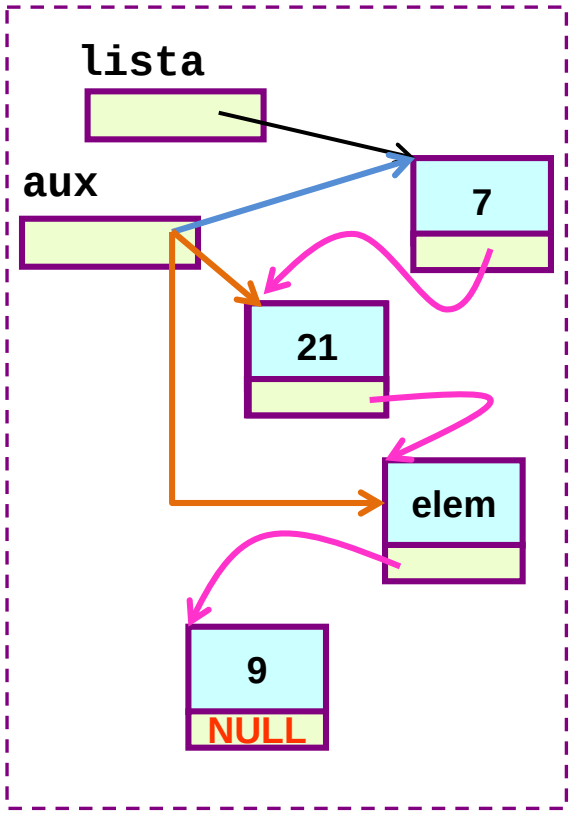
**MENTRE PENSIAMO ...** … **!=** … e` troppo ad hoc!!
Sarebbe stato meglio aver definito una funzione in grado di lavorare su TipoElem - così da non impicciarsi della struttura dati di TipoElem proprio qui,
ad esempio avremmo potuto fare una una chiamata come:

```
diversi(aux->info, elem)
```

o, meglio ancora,

```
!uguali(aux->info, elem)
```

# eliminazione di elem da lista: prima ipotesi




scansione con aux,
per cercare elem

```
aux=lista;
while (aux && !uguali(aux->info,elem) )
   aux = aux->next;
```

ora aux punta
sul nodo da eliminare

```
if (aux)
  free(aux);
```

vedi NB precedente ...:
supponendo di aver definito la funzione
**uguali(TipoElem, TipoEleme)**

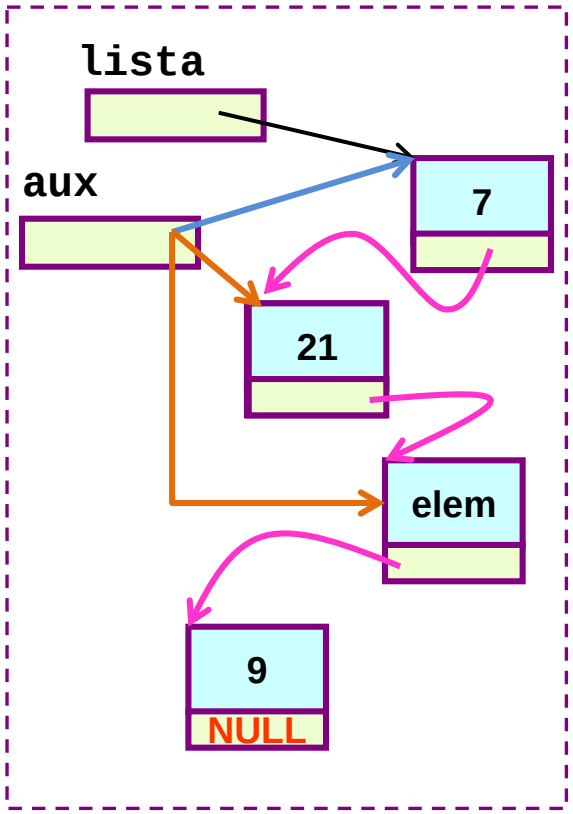
usiamo la chiamata
**!uguali(aux->info, elem)**

**no no no, un momento …**

**cosi` abbiamo impostato la soluzione, e continueremo ad usare la funzione uguali …**
**ma il modo in cui abbiamo scandito la lista non va bene …**

**qual e` il problema legato all'uso di aux? ☺**

# eliminazione di elem da lista: prima ipotesi




scansione con aux,
per cercare elem

```
aux=lista;
while (aux && !uguali(aux->info,elem) )
   aux = aux->next;
```

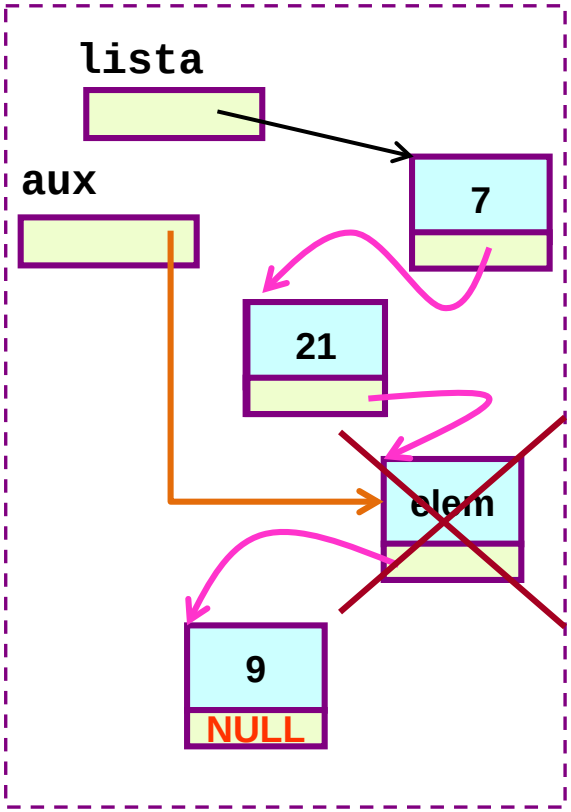
ora aux punta
sul nodo da eliminare

```
if (aux)
  free(aux);
```

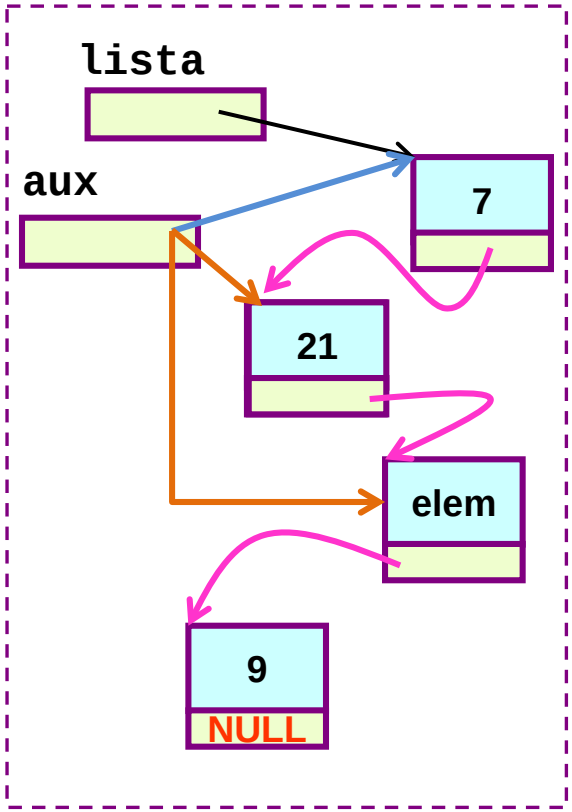
**con il che abbiamo tagliato la lista**

**NO, non e` questa la soluzione**

```
vedi NB precedente ...:
supponendo di aver definito la
funzione
uguali(TipoElem, TipoEleme)

usiamo la chiamata
          !uguali(aux->info, elem)
```



# eliminazione di elem da lista: prima ipotesi




scansione con aux,
per cercare elem

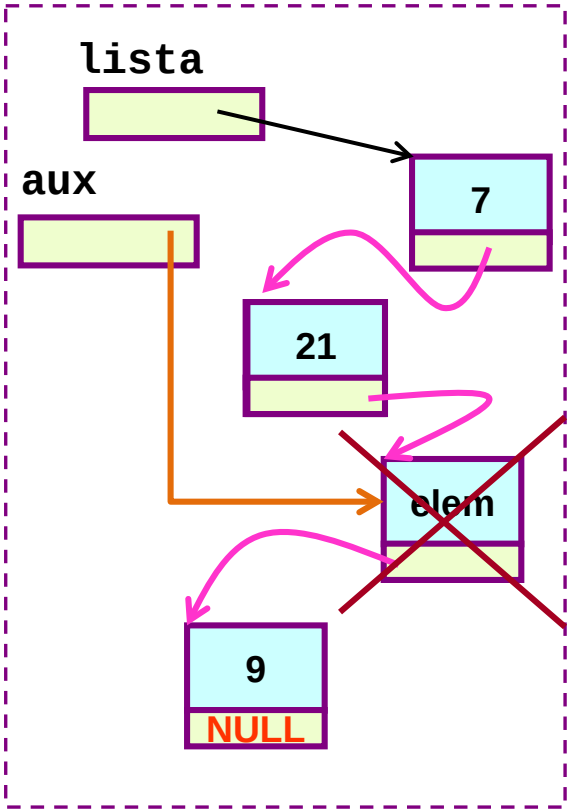
```
aux=lista;
while (aux && !uguali(aux->info,elem) )
   aux = aux->next;
```

ora aux punta
sul nodo da eliminare

```
if (aux)
   free(aux);
```

**con il che abbiamo tagliato la lista**

**NO, non e` questa la soluzione**




Dovremmo far si` che il 9 diventi il successore del 21 e poi eliminare elem … ma non possiamo perche' non abbiamo nessun puntatore al 21 …

# eliminazione di elem da lista:

seconda ipotesi per Δ

**TRE passi: A) B) C)**

**che usano due puntatori di appoggio**

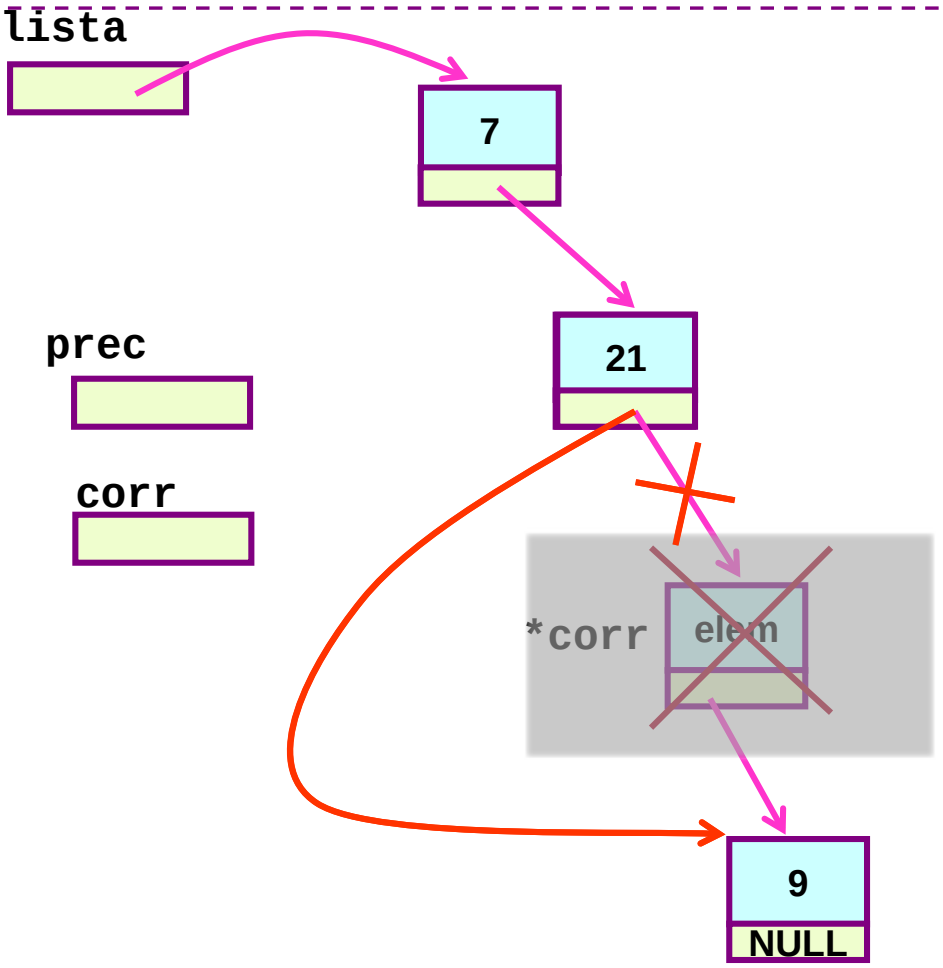
**A) scansione con corr, mantenendo sempre prec puntato sul nodo precedente quello puntato da corr** cioe` il nodo `(*corr)`

**ORA** `*corr` ≡ nodo da eliminare
`*prec` ≡ nodo precedente

**B) aggiustamento:**
**in mdo che il nodo `(*prec)` abbia come successore l'attuale successore del nodo `(*corr)`**

= **isolamento** di `*corr`

**C) free(corr)**




`questo è come lo otteniamo`

`questo e` quel che dobbiamo ottenere`

# eliminazione di elem da lista:

la seconda per Δ ipotesi e` piu` giusta

**TRE passi: A) B) C)**

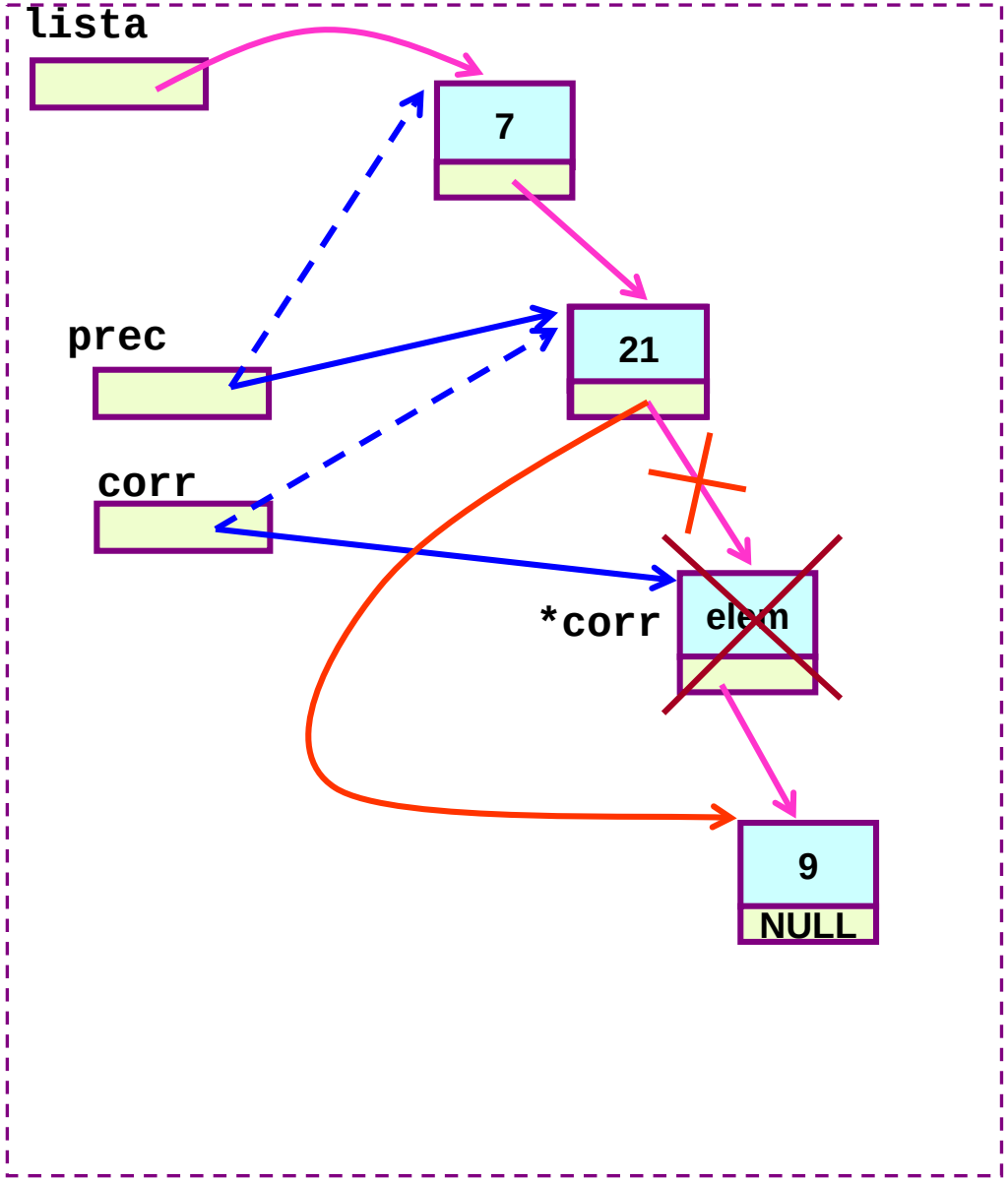
**che usano due puntatori di appoggio**

**A) scansione con corr, mantenendo sempre prec puntato sul nodo precedente quello puntato da corr (cioe` "*corr")**

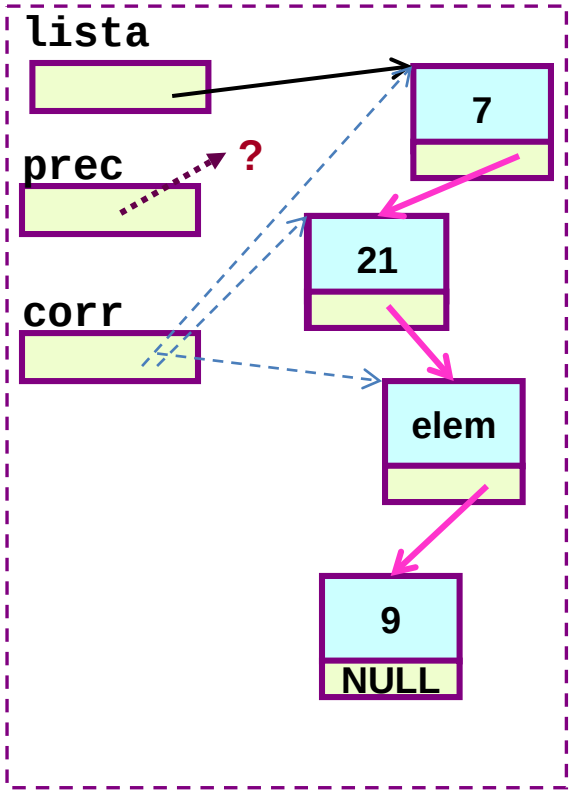
**ORA** `*corr` ≡ nodo da eliminare
`*prec` ≡ nodo precedente

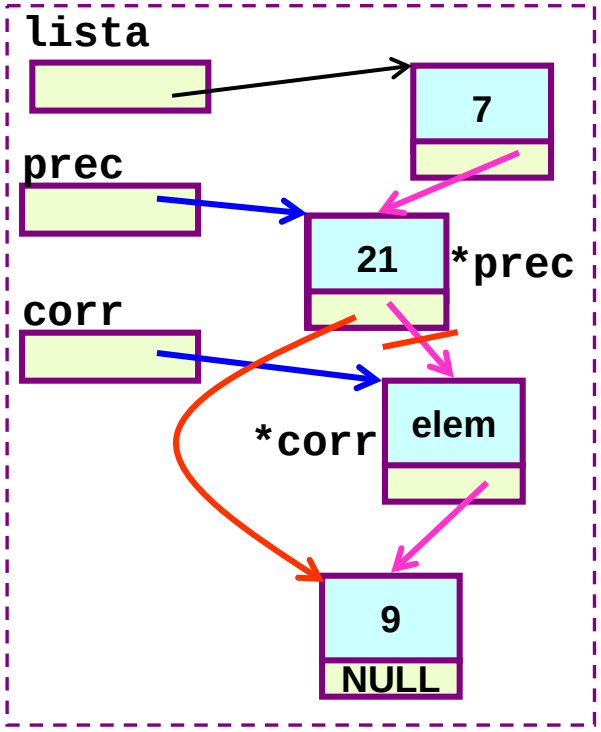
**B) aggiustamento: che `*prec` abbia come successore l'attuale successore di `*corr` = isolamento di `*corr`**

**C) free(corr)**

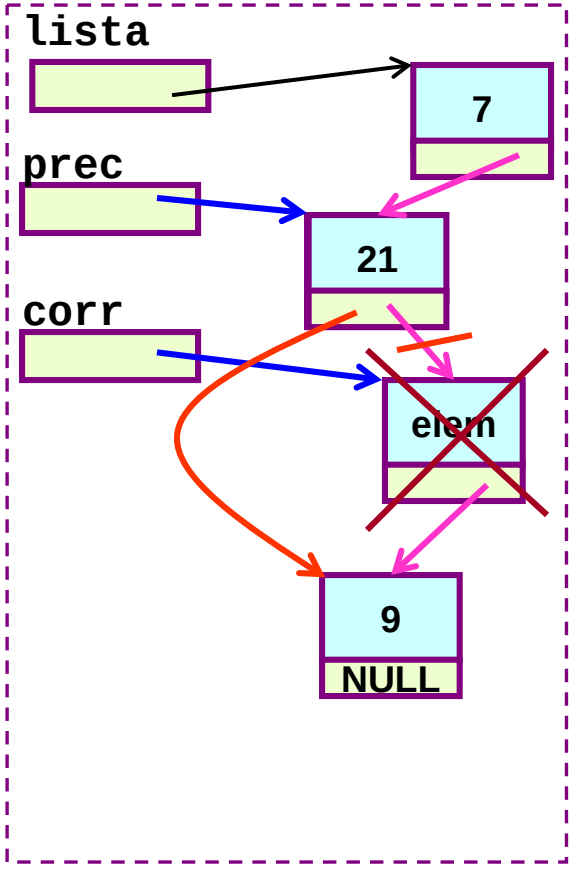


# eliminazione di elem da lista: soluzione quasi giusta per Δ




TRE passi: A) B) C)




**A) scansione con corr, mantenendo sempre prec puntato sul nodo precedente quello puntato da corr (detto "*corr")**

ORA `*corr` ≡ nodo da eliminare
`*prec` ≡ nodo precedente

**B) aggiustamento: che `*prec` abbia come successore l'attuale successore di `*corr` = isolamento di `*corr`**

**C) free(corr)**

# eliminazione di elem da lista: come "quasi"?? (eh gia`... ma da dove parte prec?)

**TRE passi: A) B) C)**

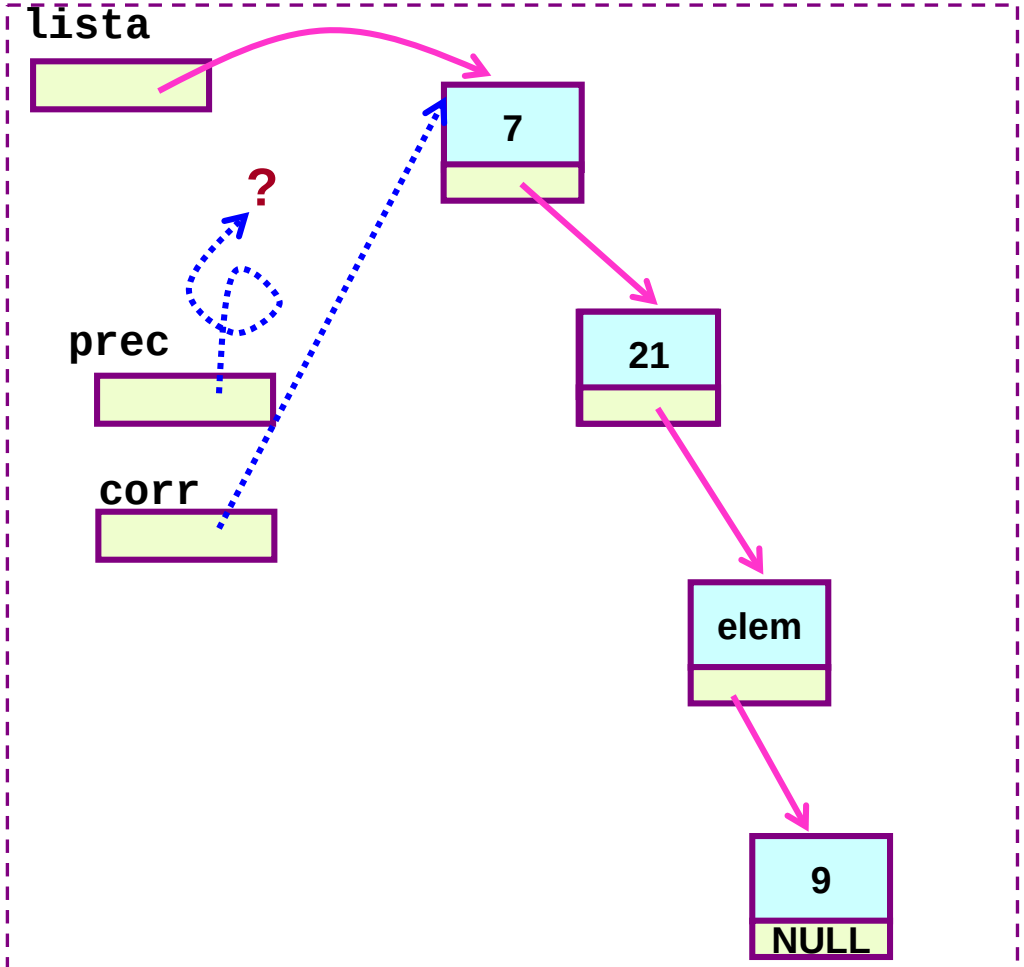
**che usano due puntatori di appoggio**

**A) scansione con corr, mantenendo sempre prec puntato sul nodo precedente quello puntato da corr (cioe` "*corr")**

**ORA** `*corr` ≡ nodo da eliminare
`*prec` ≡ nodo precedente

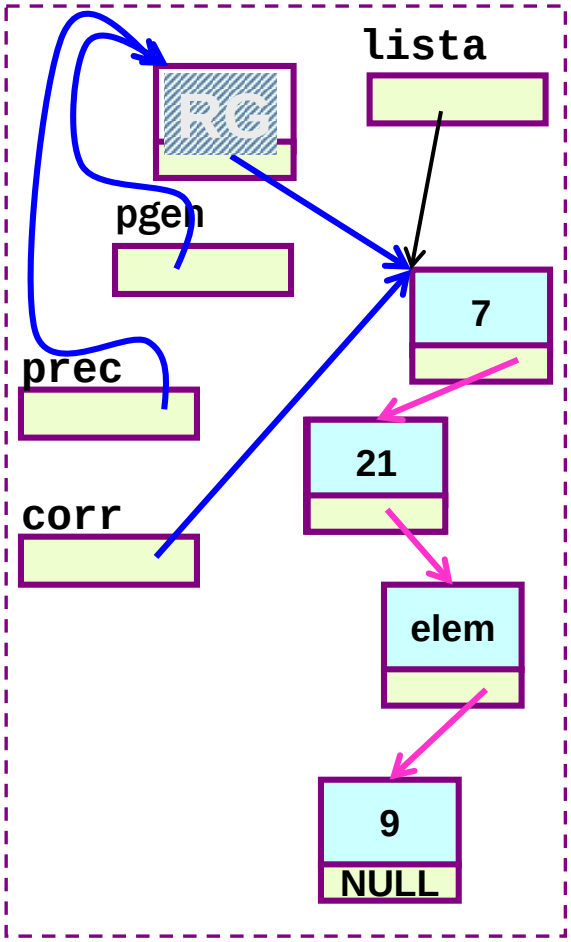
**B) aggiustamento: che `*prec` abbia come successore l'attuale successore di `*corr` = isolamento di `*corr`**

**C) free(corr)**




```
Solo che … inizialmente corr dovrebbe
puntare al primo elemento … e quindi prec
dovrebbe puntare sul …?

Come facciamo?
O programmiamo un caso particolare per il
primo elemento, oppure …
```

# eliminazione di elem da lista: verso la soluzione giusta




**INIZIALMENTE bisogna che corr punti sul primo nodo e prec sul nodo che precede il primo (!!)**

**si usa la tecnica del record generatore**

**algoritmo**

**0) RG** (alloc+posizionamento record generatore)

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

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

ORA (elem trovato) ≡ (corr<>NULL)

3) se corr<>NULL
```
prec->next = corr->next;
free(corr)
```

4) riaggiustamenti finali (posizionamento radice lista e cancellazione RG)
```
lista = pgen->next
free(pgen)
```

# eliminazione di elem da lista: tecnica del RG

**INIZIALMENTE bisogna che corr punti sul primo nodo e prec sul nodo che precede il primo (!!)**

**si usa la tecnica del record generatore**

**algoritmo**

0) RG (alloc+posizionamento record generatore)

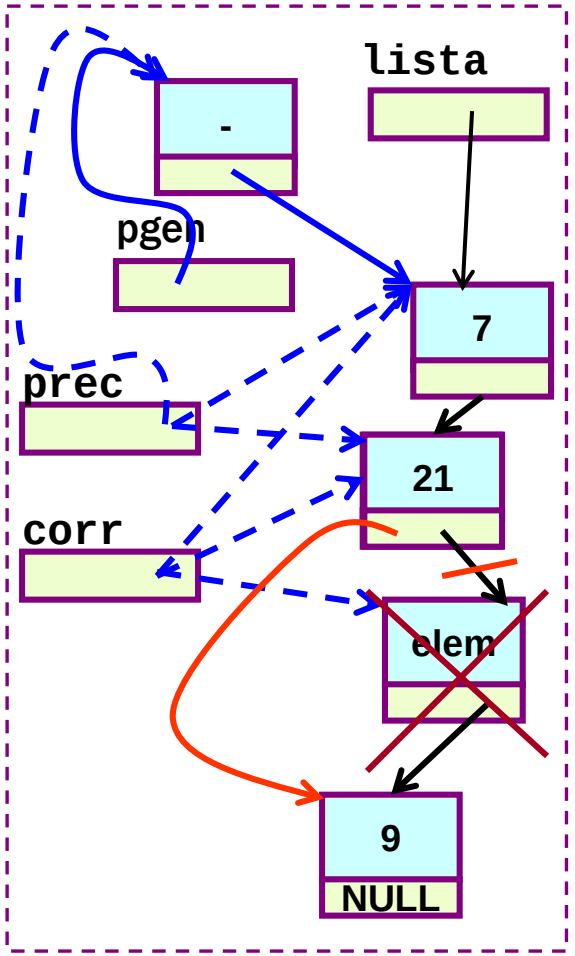
1) `prec=pgen; corr=lista;`

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

**ORA (elem trovato) ≡ (corr<>NULL)**

3) **se corr<>NULL**
`prec->next = corr->next;`
`free(corr)`

4) riaggiustamenti finali (posizionamento radice lista e cancellazione RG)
`lista = pgen->next`
`free(pgen)`



# eliminazione di elem da lista: soluzione giusta

**INIZIALMENTE bisogna che corr punti sul primo nodo e prec sul nodo che precede il primo (!!)**

**si usa la tecnica del record generatore**

**algoritmo**

**0) RG** (alloc+posizionamento record generatore)

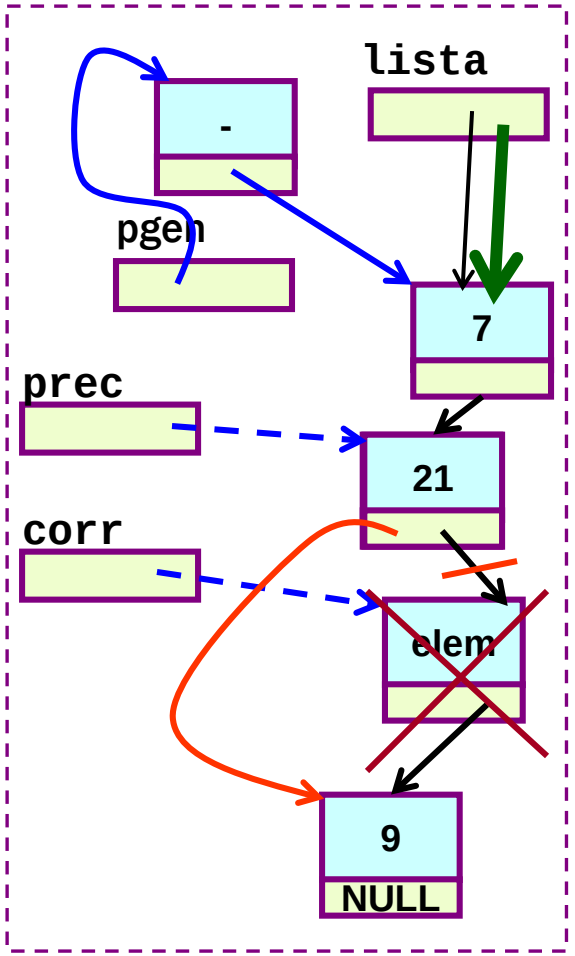
**1)** `prec=pgen; corr=lista;`

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

**ORA (elem trovato) ≡ (corr<>NULL)**

**3) se corr<>NULL**
```
prec->next = corr->next;
free(corr)
```

**4) riaggiustamenti finali (posizionamento radice lista e cancellazione RG)**
```
lista = pgen->next
free(pgen)
```

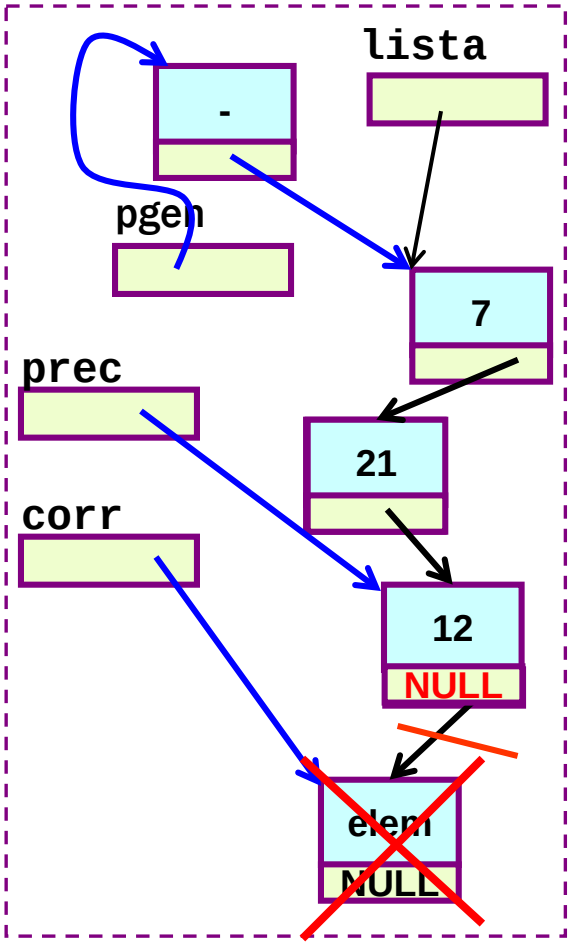


# eliminazione di `elem` da `lista`: soluzione giusta per Δ

```
/* Δ */
  pgen=malloc(sizeof(TipoNodo));                          0)
  if(!pgen)
     printf("problemi in alloc. rg ...");
  else {
       pgen->next = lista;                                1)
       prec = pgen;
       corr = lista;
       while (corr && !uguali(corr->info,elem)) {         2)
          prec = prec->next;
          corr = corr->next;
       }
       if (corr) {
          prec->next = corr->next;
          free(corr);
       }
       lista = pgen->next;                                4)
       free(pgen);
.... /* resto programma ... stampaLista ... */
```

**3) elem e` stato trovato: isolare *corr ed eliminarlo**

# eliminazione di elem da lista: casi particolari (1/3)




**l'elemento da eliminare e` l'ultimo: eliminazione da coda:**

**il ciclo**
`while (corr && (corr->info != elem))`
**termina in extremis (corr sull'ultimo) per fallimento della seconda condizione**

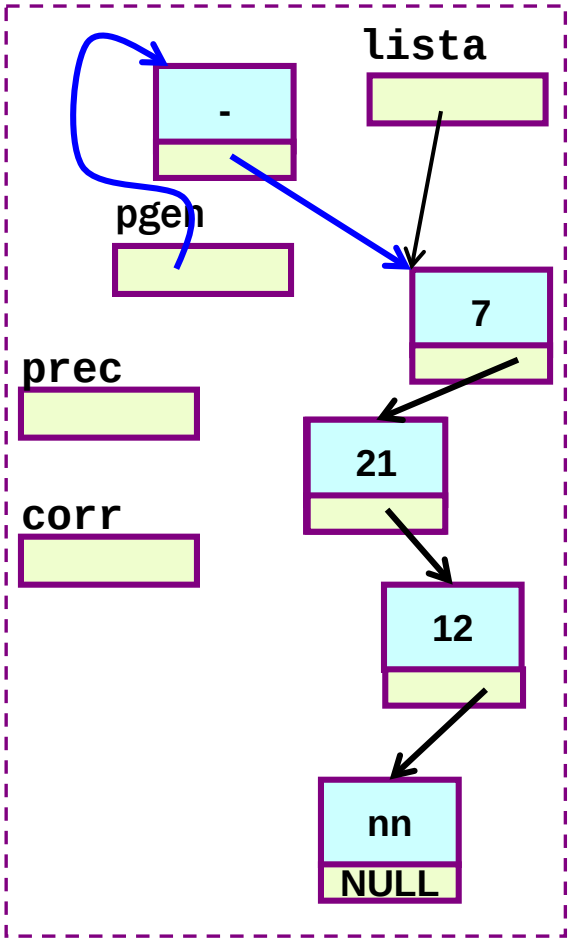
e bisogna assegnare
```
prec->next
          =
          corr->next;
          \__NULL__/
```

eliminare *corr
```
free(corr)
```

e poi
proseguire con 4)
```
lista = pgen->next
free(pgen)
```

```
pgen->next = lista;
prec = pgen;
corr = lista;
while (corr &&
      (corr->info != elem)) {
   prec = prec->next;
   corr = corr->next;
}
if (corr) {
   prec->next = corr->next;
   free(corr);
}
lista = pgen->next;
   free(pgen);
```

# eliminazione di `elem` da `lista`: casi particolari (2/3)



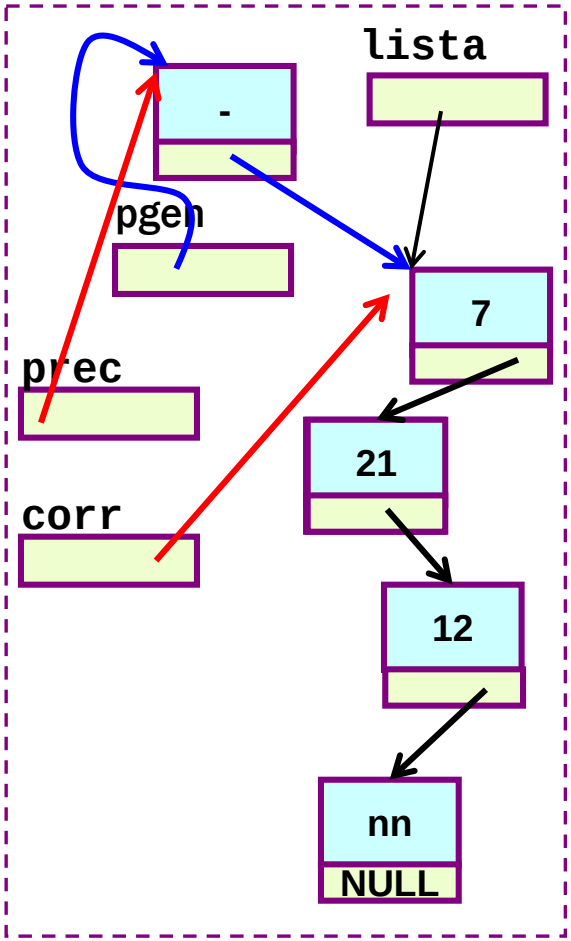

```
pgen->next = lista;
  prec = pgen;
  corr = lista;
  while (corr && (corr->info != elem)) {
    prec = prec->next;
    corr = corr->next;
  }
  if (corr) {
    prec->next = corr->next;
    free(corr);
  }
lista = pgen->next;
free(pgen)
```

**Elem e` 7: eliminazione primo**

**Se l'elemento da eliminare e` il primo in lista, il ciclo**
`while (corr && (corr->info != elem))`
**termina (subito) per fallimento della seconda condizione**
`if (corr)` **... Che si fa?**
**corr e` non nullo (punta sul primo)**
**quindi si esegue il punto 3 e poi il 4**

**che accade alla variabile lista?**
**differenza con gli altri casi? ☺**

# eliminazione di `elem` da `lista`: **casi particolari (2/3)**



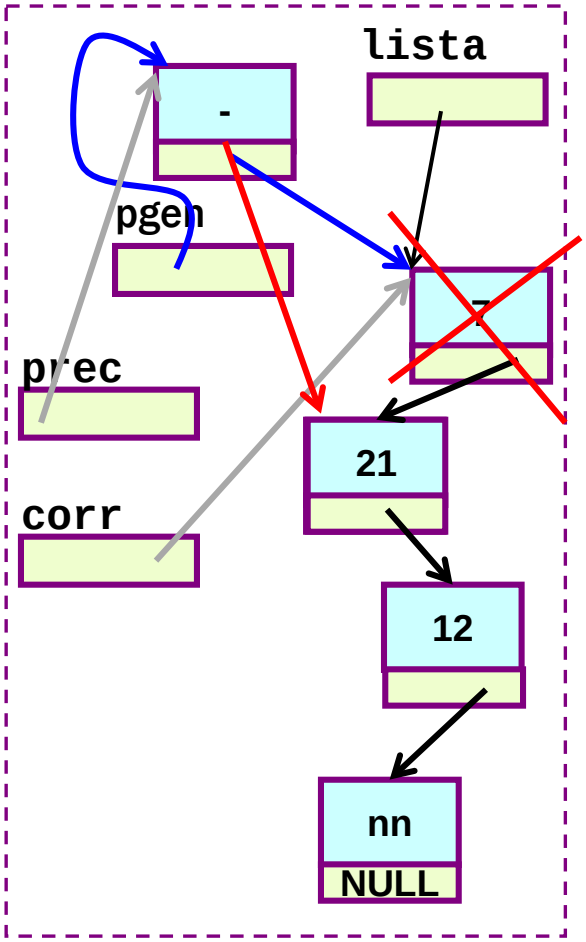

```
pgen->next = lista;
  prec = pgen;
  corr = lista;
  while (corr && (corr->info != elem)) {
    prec = prec->next;
    corr = corr->next;
  }
  if (corr) {
    prec->next = corr->next;
    free(corr);
  }
lista = pgen->next;
free(pgen)
```

**Elem e` 7: eliminazione primo**

**Se l'elemento da eliminare e` il primo in lista, il ciclo**
`while (corr && (corr->info != elem))`
**termina (subito) per fallimento della seconda condizione**
`if (corr)` **... Che si fa?**
**corr e` non nullo (punta sul primo)**
**quindi si esegue il punto 3 e poi il 4**

**che accade alla variabile lista?**
**differenza con gli altri casi? ☺**

# eliminazione di `elem` da `lista`: casi particolari (2/3)



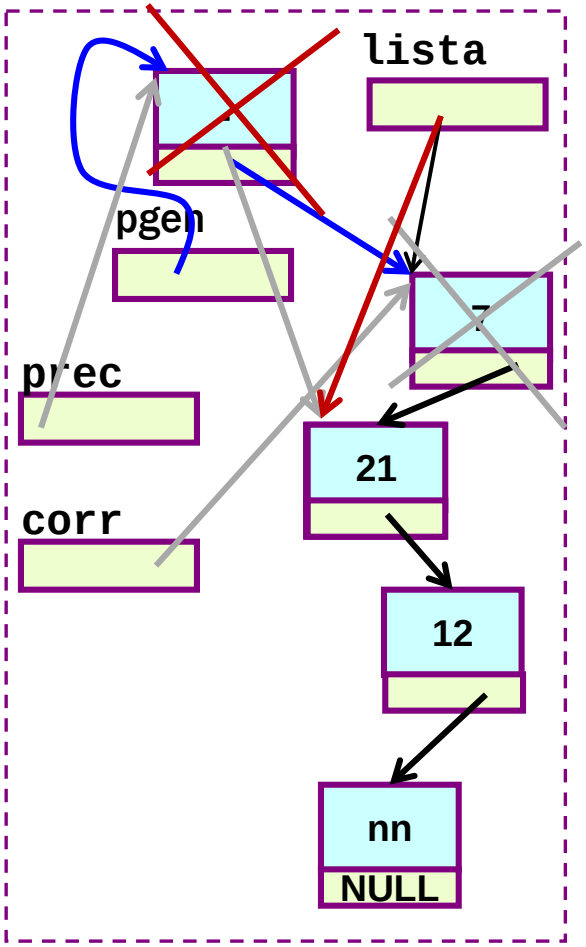

```
pgen->next = lista;
prec = pgen;
corr = lista;
while (corr && (corr->info != elem)) {
   prec = prec->next;
   corr = corr->next;
}
if (corr) {
   prec->next = corr->next;
   free(corr);
}
lista = pgen->next;
free(pgen)
```

**Elem e` 7: eliminazione primo**

**Se l'elemento da eliminare e` il primo in lista, il ciclo**
`while (corr && (corr->info != elem))`
**termina (subito) per fallimento della seconda condizione**
`if (corr)` **... Che si fa?**
**corr e` non nullo (punta sul primo)**
**quindi si esegue il punto 3 e poi il 4**

**che accade alla variabile lista?**
**differenza con gli altri casi? ☺**

# eliminazione di `elem` da `lista`: casi particolari (2/3)




```
pgen->next = lista;
prec = pgen;
corr = lista;
while (corr && (corr->info != elem)) {
  prec = prec->next;
  corr = corr->next;
}
if (corr) {
  prec->next = corr->next;
  free(corr);
}
```

**lista = pgen->next;**

**free(pgen)**

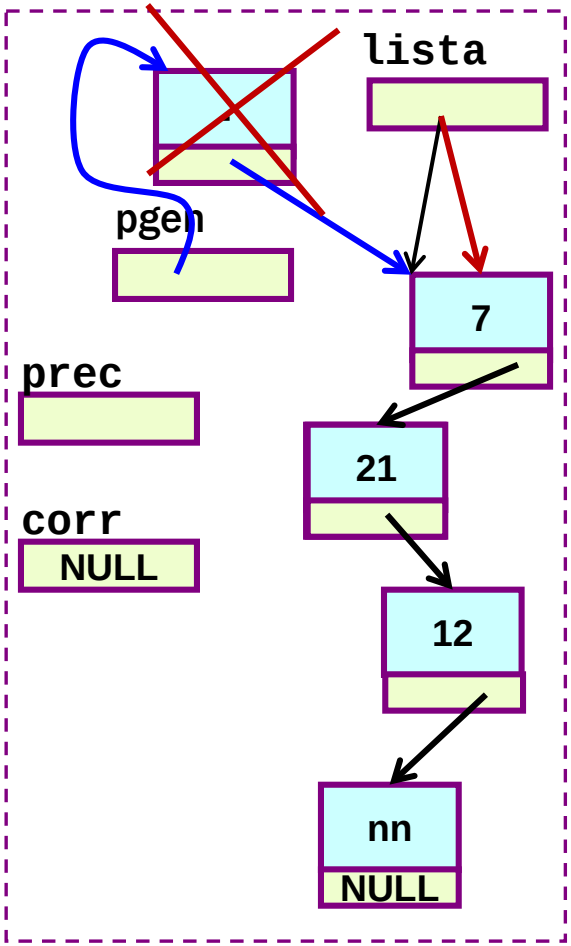
**Elem e` 7: eliminazione primo**

**Se l'elemento da eliminare e` il primo in lista, il ciclo**
`while (corr && (corr->info != elem))`
**termina (subito) per fallimento della seconda condizione**
`if (corr)` **... Che si fa?**
**corr e` non nullo (punta sul primo)**
**quindi si esegue il punto 3 e poi il 4**

**che accade alla variabile lista?**
**differenza con gli altri casi? ☺**

**Yes, `lista` cambia** e poi punta sull'ex-secondo)

# eliminazione di `elem` da `lista`: casi particolari (3/3)




```
pgen->next = lista;
   prec = pgen;
   corr = lista;
   while (corr && (corr->info != elem)) {
      prec = prec->next;
      corr = corr->next;
   }
   if (corr) {
      prec->next = corr->next;
      free(corr);
   }
lista = pgen->next;
free(pgen)
```

**l'elemento da eliminare non c'e` in lista: il ciclo**
`while (corr && (corr->info != elem))`
**termina per fallimento della prima condizione**

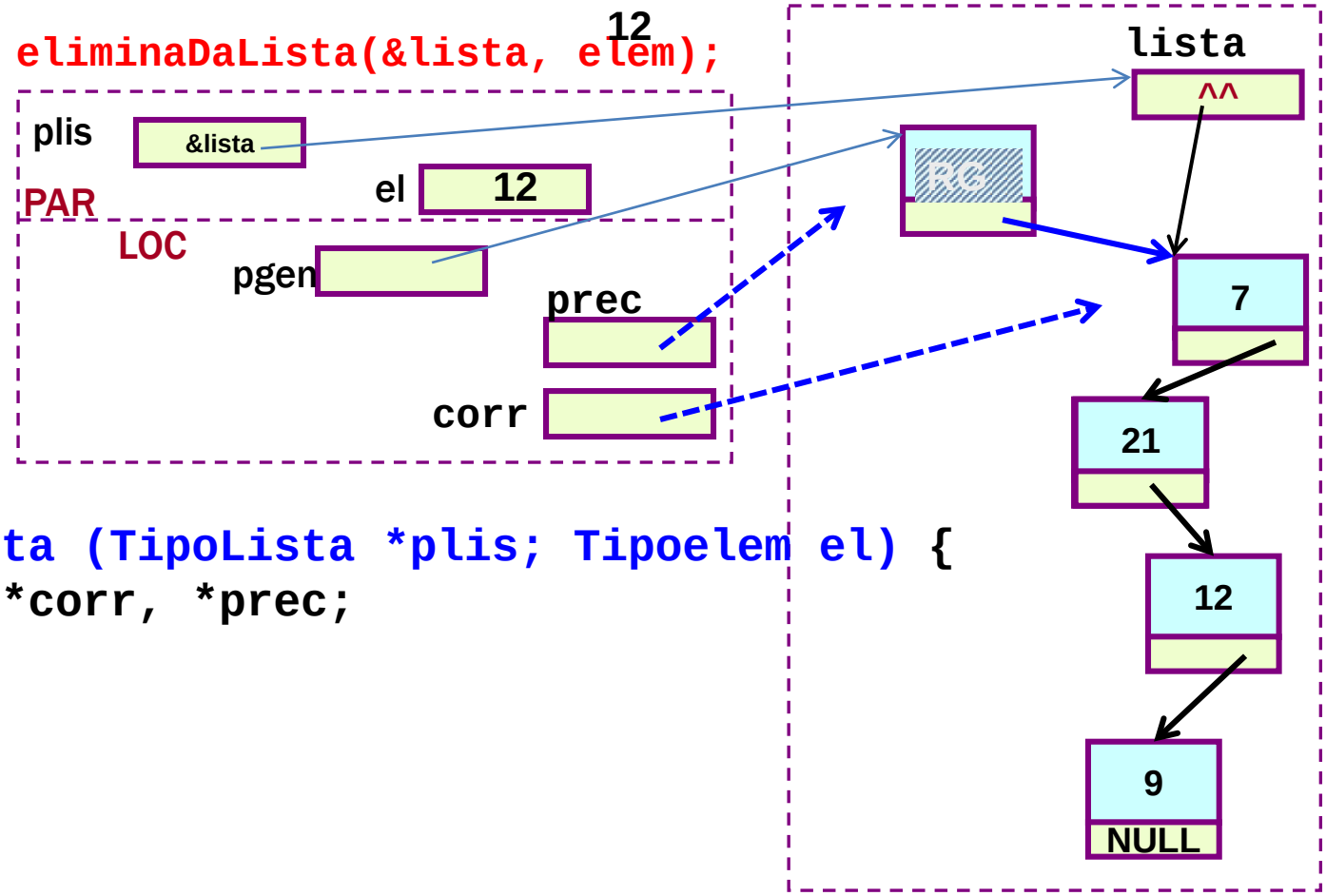
`if (corr)` **... tsk corr e` NULL ... niente da fare, tranne che proseguire con 4)**

# Eliminazione da una lista: mediante funzione e side effect (0/3)

```
...typedef int TipoElem;...

int main() {
 TipoLista lista;
 TipoElem elem;
...
  /* costruzione lista */
...
  leggiElem(&elem);  /* l'el. da eliminare */
...
  eliminaDaLista(&lista, elem); /* eliminazione */
...
  stampaLista(lista);
...
...
return 0;
}
```

NB nel caso in cui il nodo da eliminare è il primo, cambia il valore di lista: quindi dobbiamo gestire la funzione in modo che, possa provocare un effetto collaterale sul puntatore all'inizio della lista (cioè dobbiamp [assare l'indirizzo di lista), in modo che lista possa cambiare da prima a dopo dell'esecuzione della funzione

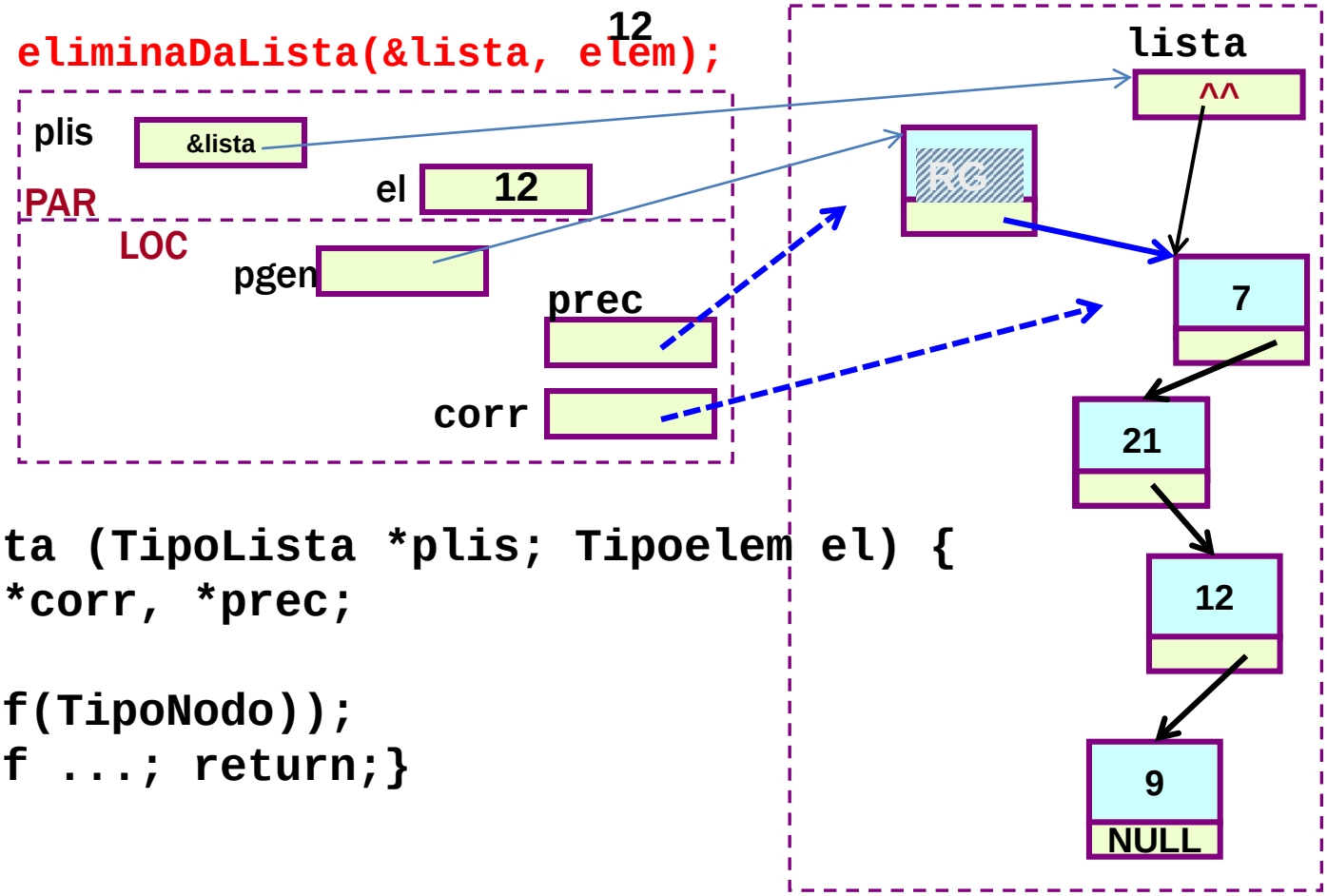
# Eliminazione da una lista: mediante funzione e side effect (1/3)




```
void eliminaDaLista (TipoLista *plis; Tipoelem el) {
 TipoNodo *pgen, *corr, *prec;
```

```
}
```

# Eliminazione da una lista: mediante funzione e side effect (1/3)



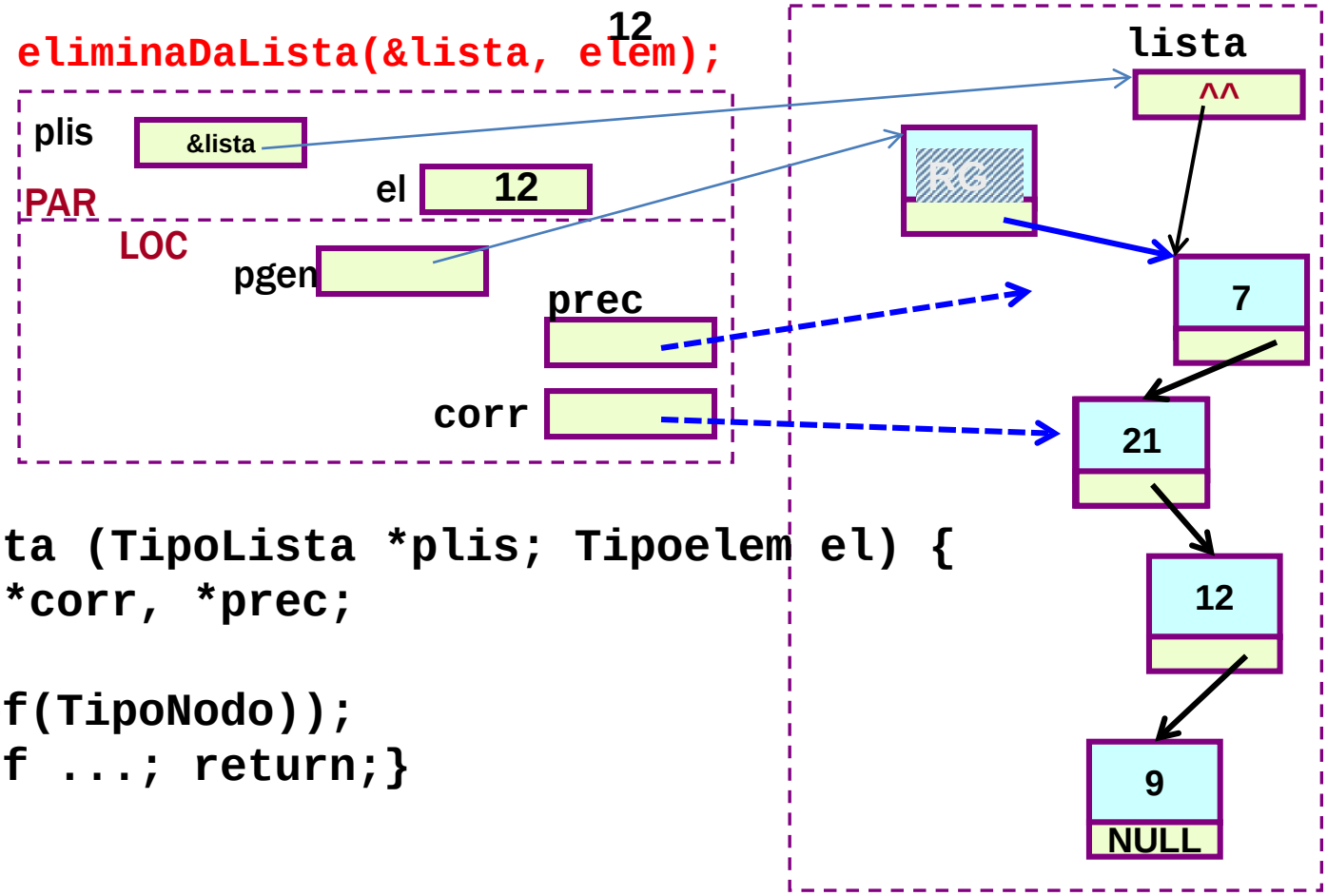

```
void eliminaDaLista (TipoLista *plis; Tipoelem el) {
 TipoNodo *pgen, *corr, *prec;

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

prec=pgen;
corr = pgen->next = *plis;

while (corr && !uguali(corr->info, el)) {
  corr = corr->next;
  prec = prec->next;
}
```

# Eliminazione da una lista: mediante funzione e side effect (1/3)

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

prec parte puntando su *pgen e finisce puntando su 21;

corr parte puntando sul 7 e finisce puntando sul 12.



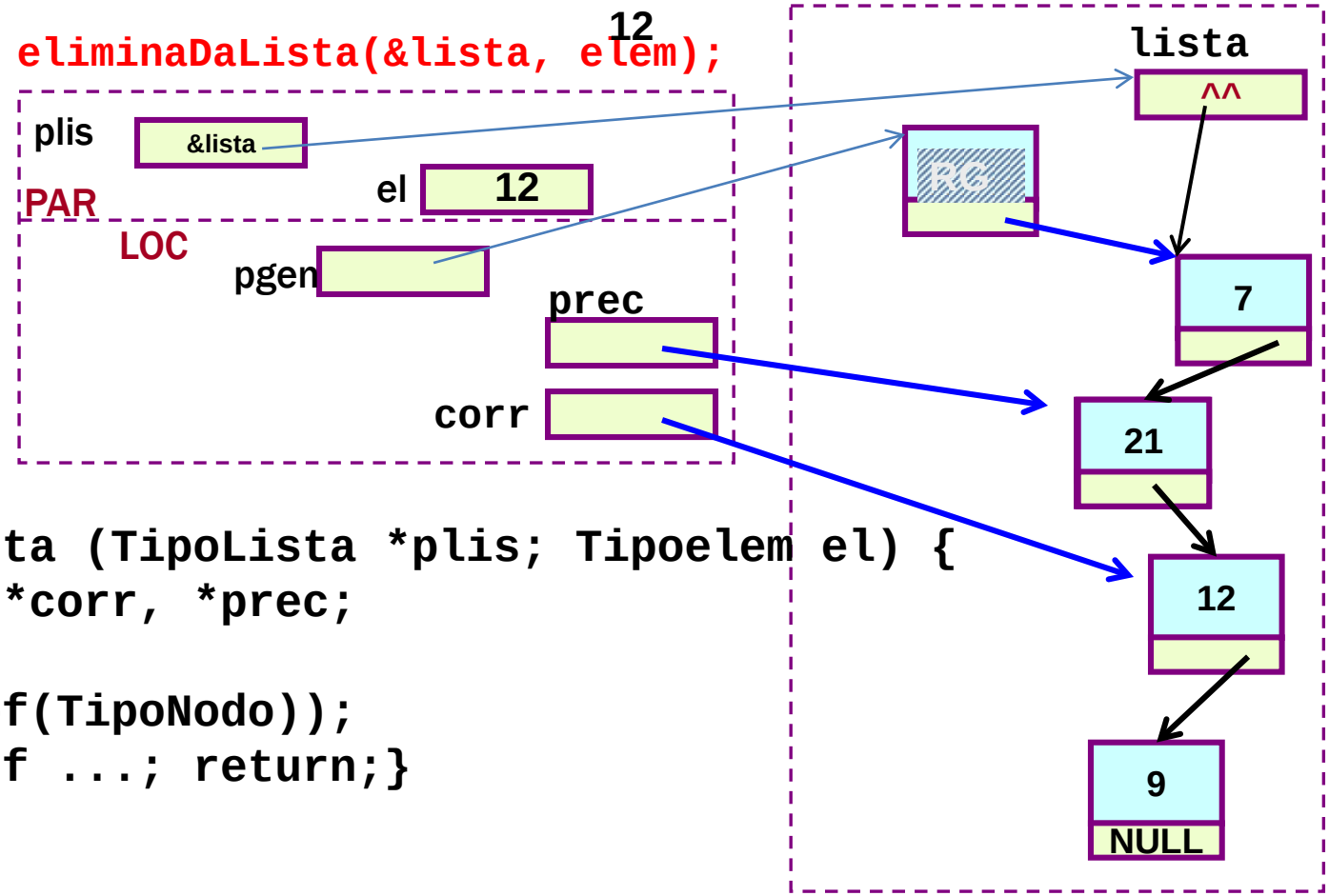

```
void eliminaDaLista (TipoLista *plis; Tipoelem el) {
 TipoNodo *pgen, *corr, *prec;

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

prec=pgen;
corr = pgen->next = *plis;

while (corr && !uguali(corr->info, el)) {
  corr = corr->next;
  prec = prec->next;
}
```

# Eliminazione da una lista: mediante funzione e side effect (1/3)

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

prec parte puntando su *pgen e finisce puntando su 21;

corr parte puntando sul 7 e finisce puntando sul 12.



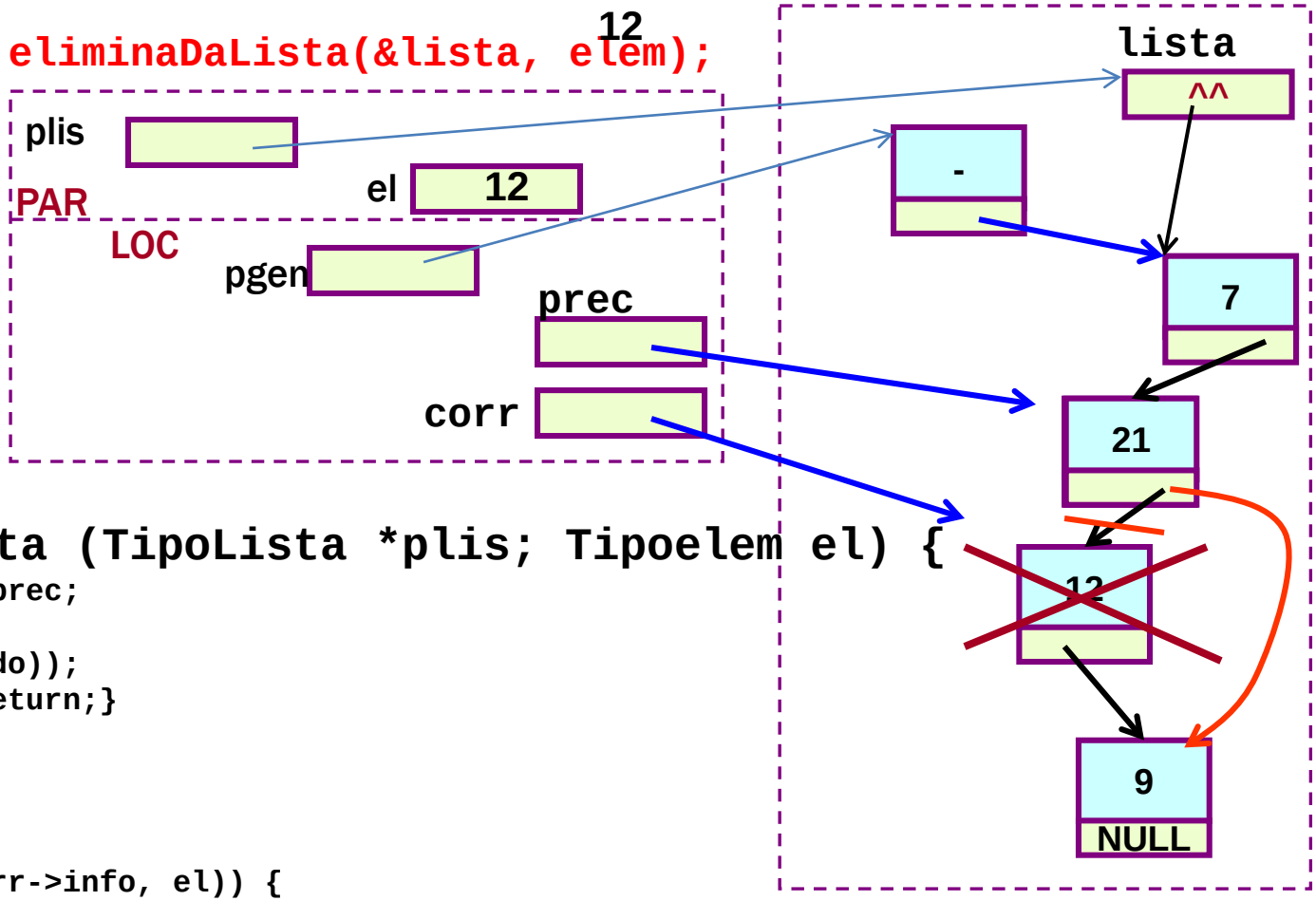

```
void eliminaDaLista (TipoLista *plis; Tipoelem el) {
 TipoNodo *pgen, *corr, *prec;

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

prec=pgen;
corr = pgen->next = *plis;

while (corr && !uguali(corr->info, el)) {
  corr = corr->next;
  prec = prec->next;
}
```

# Eliminazione da una lista: mediante funzione e side effect (2/3)

prec parte puntando su *pgen e finisce puntando su 21;

corr parte puntando sul 7 e finisce puntando sul 12.

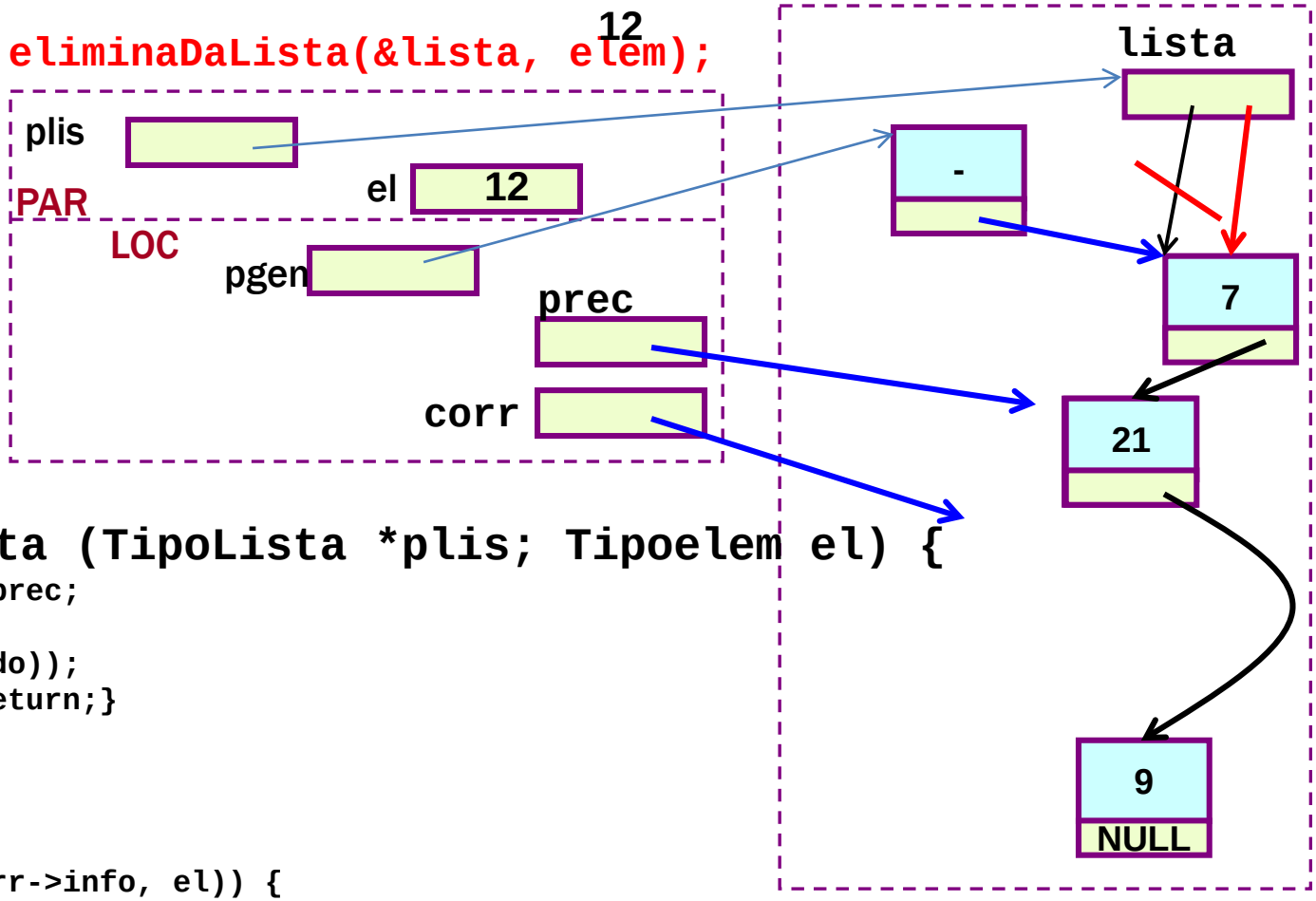
il nodo puntato da corr (*corr) viene isolato ed eliminato




```
void eliminaDaLista (TipoLista *plis; Tipoelem el) {
 TipoNodo *pgen, *corr, *prec;

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

prec=pgen;
corr=pgen->next = *plis;

while (corr && !uguali(corr->info, el)) {
  corr = corr->next;
  prec = prec->next;
}
if (corr) { prec->next = corr->next; free(corr); }
*plis = pgen->next;
free(pgen);
return; }
```

# Eliminazione da una lista: mediante funzione e side effect (3/3)

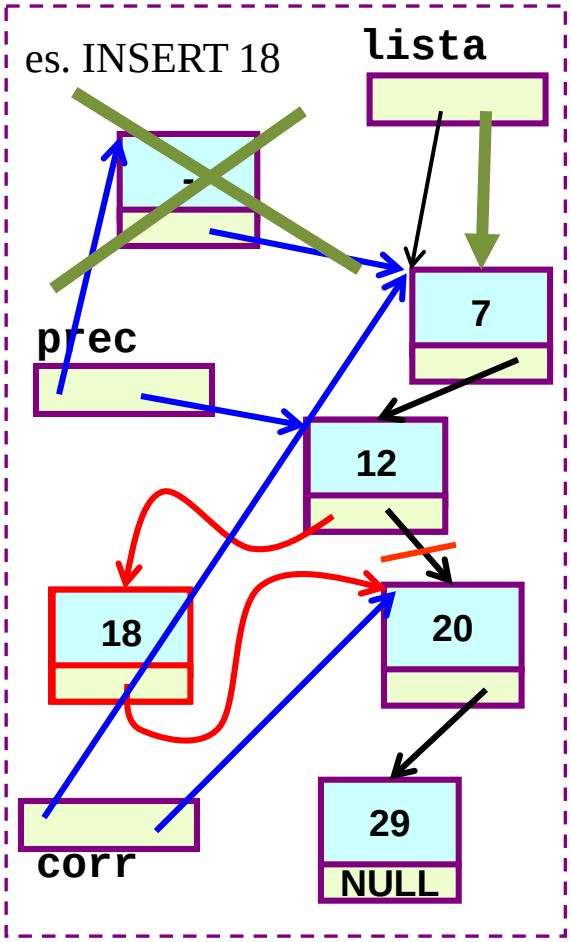
*plis cioe` lista viene assegnata con l'indirizzo del nodo successore di *pgen (nell'esempio e` sempre il 7, ma se il 7 fosse stato eliminato sarebbe il 21 e il valore iniziale di lista sarebbe giustamente cambiato)




```
void eliminaDaLista (TipoLista *plis; Tipoelem el) {
 TipoNodo *pgen, *corr, *prec;

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

prec=pgen;
corr=pgen->next = *plis;

while (corr && !uguali(corr->info, el)) {
  corr = corr->next;
  prec = prec->next;
}
if (corr) { prec->next = corr->next; free(corr); }
*plis = pgen->next;
free(pgen);
return; }
```

poi, dopo il riassestamento di *plis (lista) il Record Generatore viene eliminato e la chiamata della funzione termina

# Inserimento in lista ordinata




Si suppone che la lista sia *ordinata* (cioè ***ogni nodo sia*** < ***del successore*** - quindi assumiamo anche che esista una relazione d'ordine "<" sui valori del tipo TipoElem, realizzata da una funzione `minore(TipoElem, TipoElem)`

**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"**

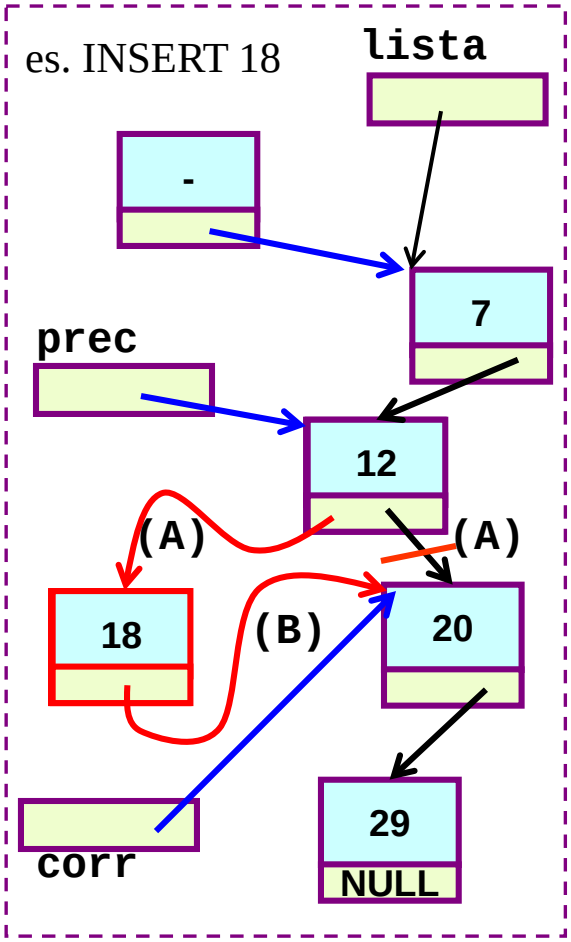
**ORA *prec=predecessore e *corr=successore del nuovo nodo**

**3) creazione e inserimento nuovo nodo**

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

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

# Inserimento in lista ordinata: inserimento del nuovo nodo




```
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)
```

# Inserimento ordinato in lista - caso della funzione main()

```
int main() {
 TipoLista lista;
 TipoNodo *pgen, *corr, *prec, *nuovo;
 TipoElem el;
...                    /* costruzione lista */
  printf("elemento da inserire? ");
  leggiElem(&el);
...
 /* 1 */
  pgen = malloc(sizeof(TipoNodo));
  if !pgen ...  (non si puo` eliminare … non resta che  uscire o tornare al menu` o …)
  pgen->next = lista;
  prec = pgen;
  corr = lista;
 /* 2 */
```

# Inserimento ordinato in lista - caso della funzione main()

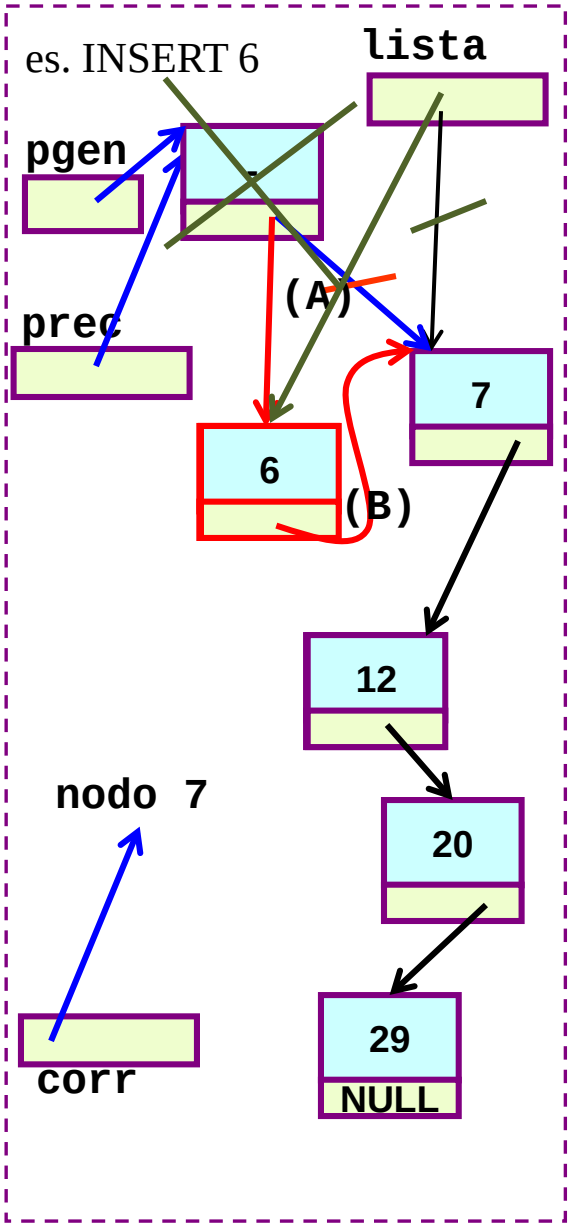
```
int main() {
 TipoLista lista;
 TipoNodo *pgen, *corr, *prec, *nuovo;
 TipoElem el;
...                    /* costruzione lista */
  printf("elemento da inserire? ");
  leggiElem(&el);
...
 /* 1 */
  pgen = malloc(sizeof(TipoNodo));
  if !pgen ...  (non si puo` eliminare … non resta che  uscire o tornare al menu` o …)
  pgen->next = lista;
  prec = pgen;
  corr = lista;
 /* 2 */
  while (corr && minore(corr->info, el)) {
     prec = corr;
     corr= corr->next;
  } ...
```

# Inserimento ordinato in lista - caso della funzione main()

```
...
  /* 1 */
  pgen = malloc(sizeof(TipoNodo));
  if !pgen ...
  pgen->next = lista;
  prec = pgen;
  corr = lista;
 /* 2 */
  while (corr && minore(corr->info, el)) {
        prec = corr;
     corr= corr->next;
  } ...
  /* 3 */
   nuovo = malloc(sizeof(TipoNodo));
   if (nuovo) {
      assegnaEl(&(nuovo->info), el);
      prec->next = nuovo;                /* (A) */
      nuovo->next = corr;                /* (B) */
   } else  printf(…!!!allocazione nuovo nodo non possibile …);

  /* 4 */

   lista = pgen->next;

   free(pgen);
                                  ... stampaLista ... return 0; }
```

funzione che assegna ad una variabile di tipo TipoInfo, un valore di tipo TipoInfo. La variabile e` passata tramite indirizzo per permettere di modificarla

# Inserimento ordinato in lista – eseguito nella funzione main(): **casi particolari**




**l'elemento da inserire sara` il primo (inserimento in testa)**

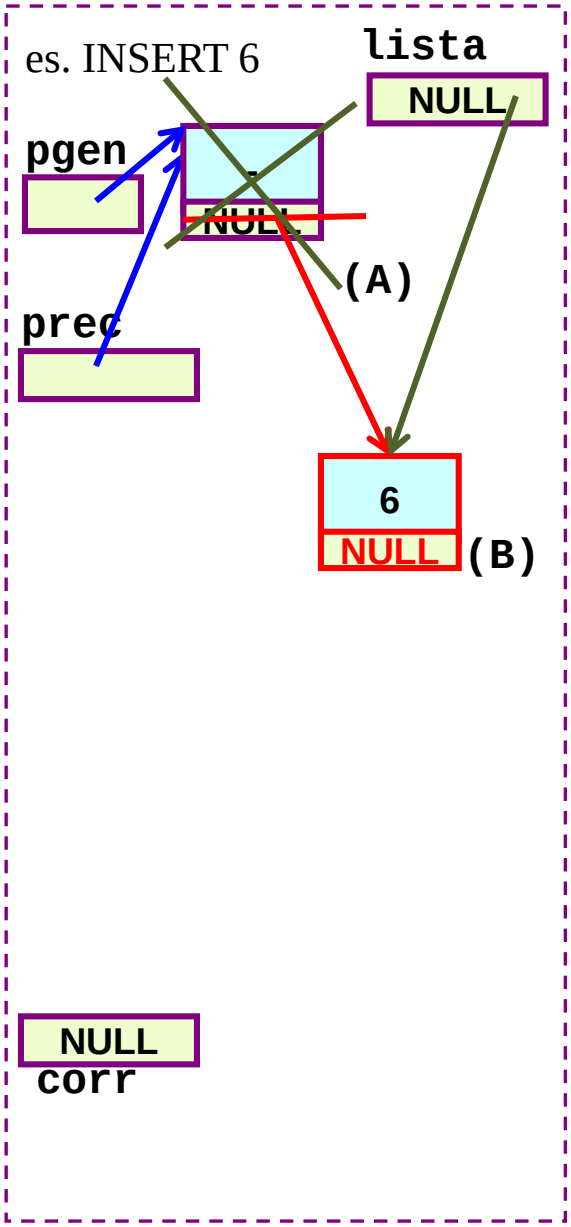
**il ciclo**
`while (corr && (corr->info < elem))`
**termina SUBITO per fallimento della seconda condizione**

```
nuovo = malloc ...

nuovo->info = ...

prec->next = nuovo (A)

nuovo->next = corr (B)

...
lista = pgen->next

free pgen
```

NB `lista` cambia da prima a dopo l'inserimento

**algoritmo**
1) RG; `prec = pgen; corr=`
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)`

# Inserimento ordinato in lista - caso della funzione main(): casi particolari




**inserimento in lista vuota**

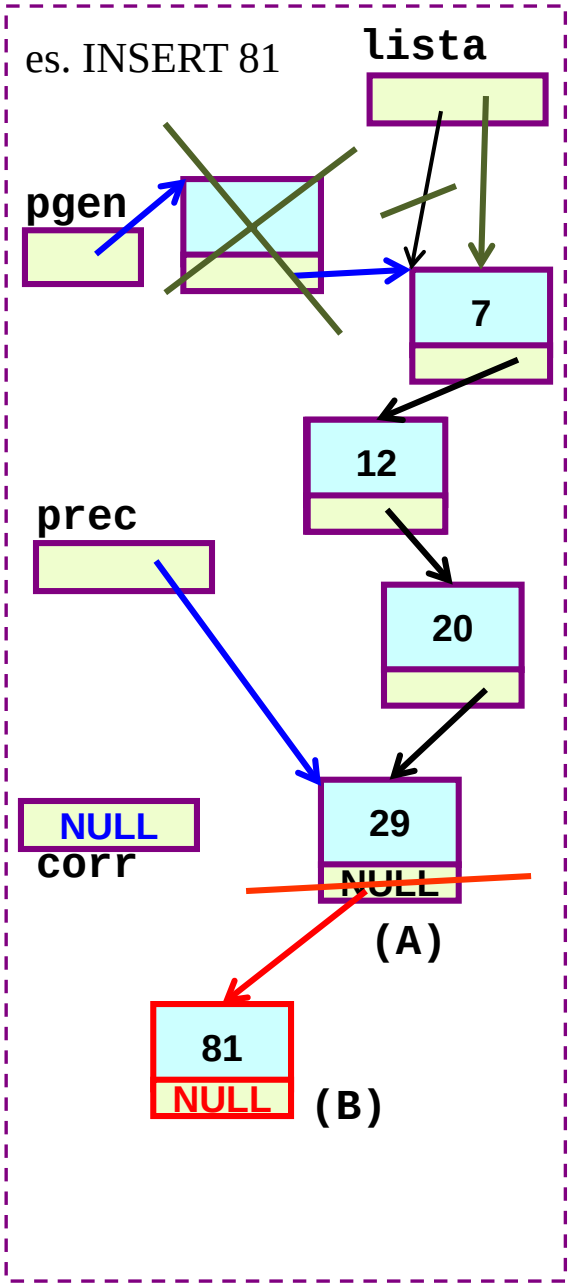
**il ciclo**
`while (corr && (corr->info < elem))`
**termina SUBITO per fallimento della prima condizione**

```
nuovo = malloc ...

nuovo->info = ...

prec->next = nuovo (A)

nuovo->next = corr (B)

...
lista = pgen->next

free pgen
```

NB `lista` cambia da prima a dopo l'inserimento

**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)`

# Inserimento ordinato in lista - caso della funzione main(): casi particolari




**inserimento in coda (il nuovo elemento sara` l'ultimo in lista)**

**il ciclo**

```
while (corr && (corr->info < elem))
```

**termina dopo aver scorso tutta la lista per fallimento della prima condizione**

```
nuovo = malloc ...

nuovo->info = ...

prec->next = nuovo (A)

nuovo->next = corr (B)

...
lista = pgen->next

free pgen
```

NB `lista` non cambia da prima a dopo l'inserimento

**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)
```