mercoledì 7 marzo 2012

```
int f(int & x, int* y){x=*y; return x;} //1
int & f(int x, int& y){y=x; return y;}//2
int*  f(int* x, const int *y){*x=*y; return x;}//3
main()
{
  int A[]={10,20,30,40,50}, i=1, c=1;
  while(i<3)
        switch(c)
        {
        case 0: f(A[1],A[2])=A[i]; c++; break; //a
        case 1: A[i]=*f(A+2,A+3); c++;//b
        case 3: A[i]=*f(&A[c],A); i++; break;//c
        default: c--; i++;
        }
  cout<<A[0]<<' '<<A[1]<<' '<<A[2]<<' '<<A[3]<<'
'<<A[4]<<endl;
}
```

Programmazione: Data una lista L, i cui nodi hanno tipo struct nodo{char info; nodo* next;}; ed un array di caratteri P di lunghezza dimP, si vuole cercare P[0] in un nodo di L, eliminare questo nodo da L, poi cercare P[1] in un successivo nodo di L, eliminare questo nodo e continuare così fino a che sia possibile. Si osservi che il nodo eliminato per P[0], deve precedere in L quello eliminato per P[1] e così via per i successivi.
Si chiede di scrivere una funzione ricorsiva
nodo*del(nodo*L, char*P, int dimP); che soddisfi le seguenti pre- e postcondizione. Nella postcondizione, la scrittura L-P indica la lista ottenuta da L eliminando i nodi che corrispondono agli elementi di P, come spiegato nell'esempio precedente.

PRE=(L è una lista event. vuota, P è un array di char di dimP elementi, dimP>=0)
POST=(restituisce la lista L-P)
Dare la dimostrazione induttiva del fatto che la funzione è corretta rispetto a PRE e POST.

nodo*del(nodo*L, char*P, int dimP);

del deve cambiare la lista L che riceve e restituire la lista L-P modificata

PRE=(L è una lista event. vuota, P è un array di char di dimP elementi, dimP>=0)

indica casi base:
P vuoto e L vuoto ➔ return L è corretto rispetto a POST ?

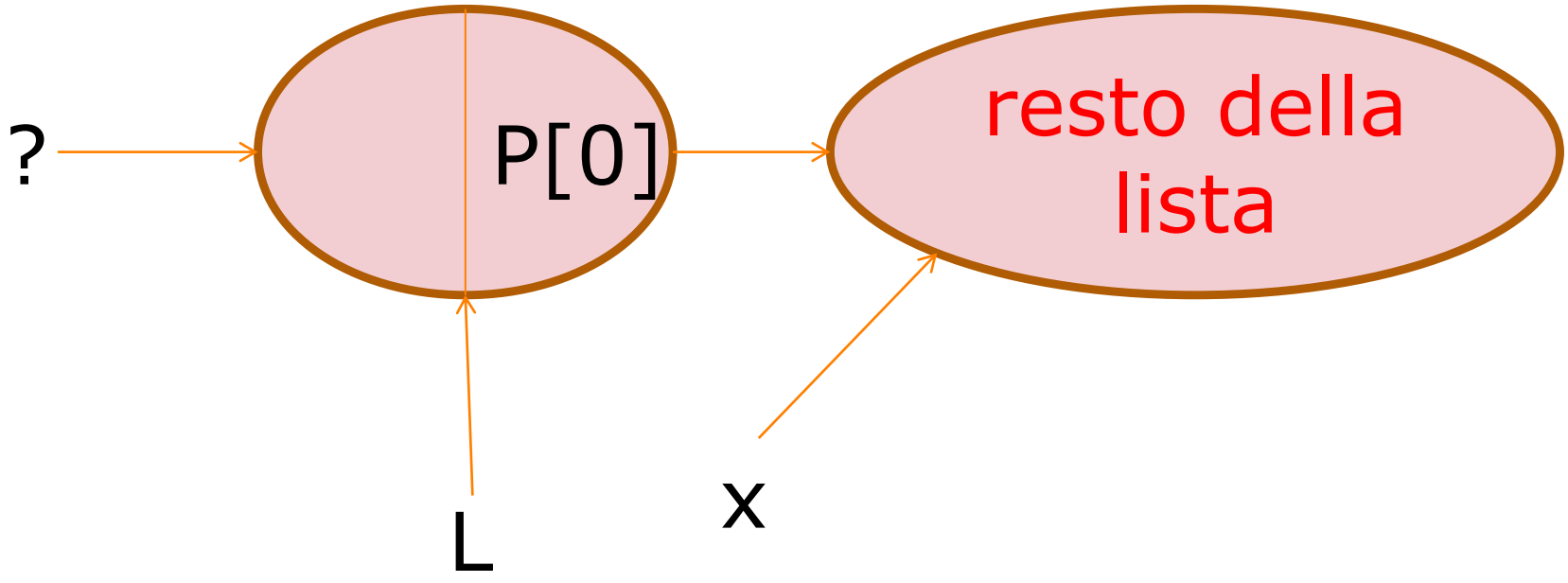
- se L è vuoto L-P è L
- se P è vuoto L-P è L

quindi restituire L soddisfa la POST

Caso ricorsivo: L non è vuoto e P non è vuoto

dobbiamo vedere se il primo modo fa match con P[0] oppure no

--L->info==P[0] : devo eliminare L dalla lista e continuare il match tra il resto della lista e il **resto** del pattern.
--L->info != P[0] devo continuare il match tra il resto della lista e lo **stesso** pattern

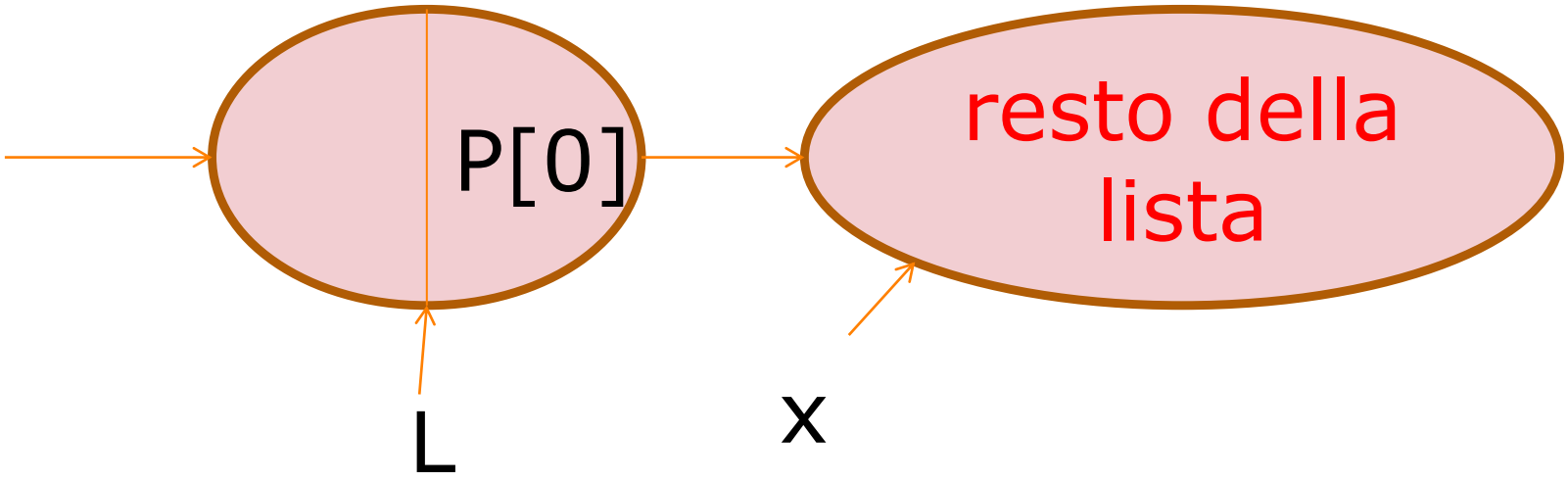



```
nodo*x=L->next;
delete L;
return del(x,P+1,dimP-1);
```

vale la PRE_ric?
si, L lista event. vuota, dimP>=0

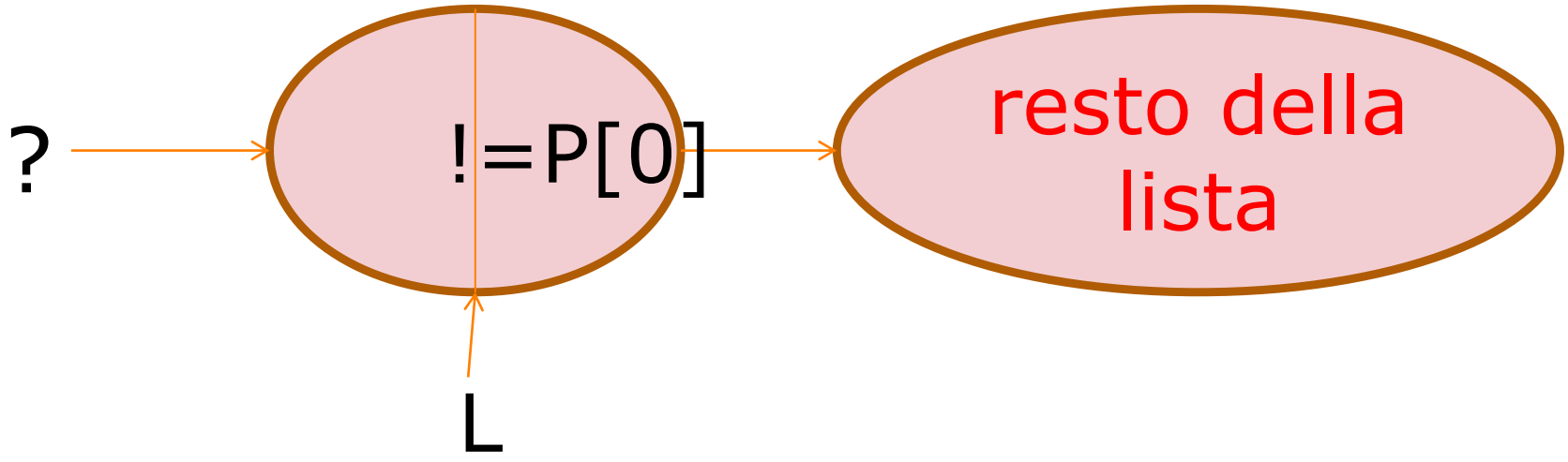
restituiamo del(resto della lista, resto del pattern) è corretto rispetto alla POST?

Essa richiede di restituire L-P



L-P significa eliminare il primo nodo e poi togliere P[1..dimP-1] dal resto della lista,

L-P=del(resto L, resto P)




```
del(L->next, P, dimP);
return L;
```

```
L->next=del(L->next, P, dimP);
return L;
```

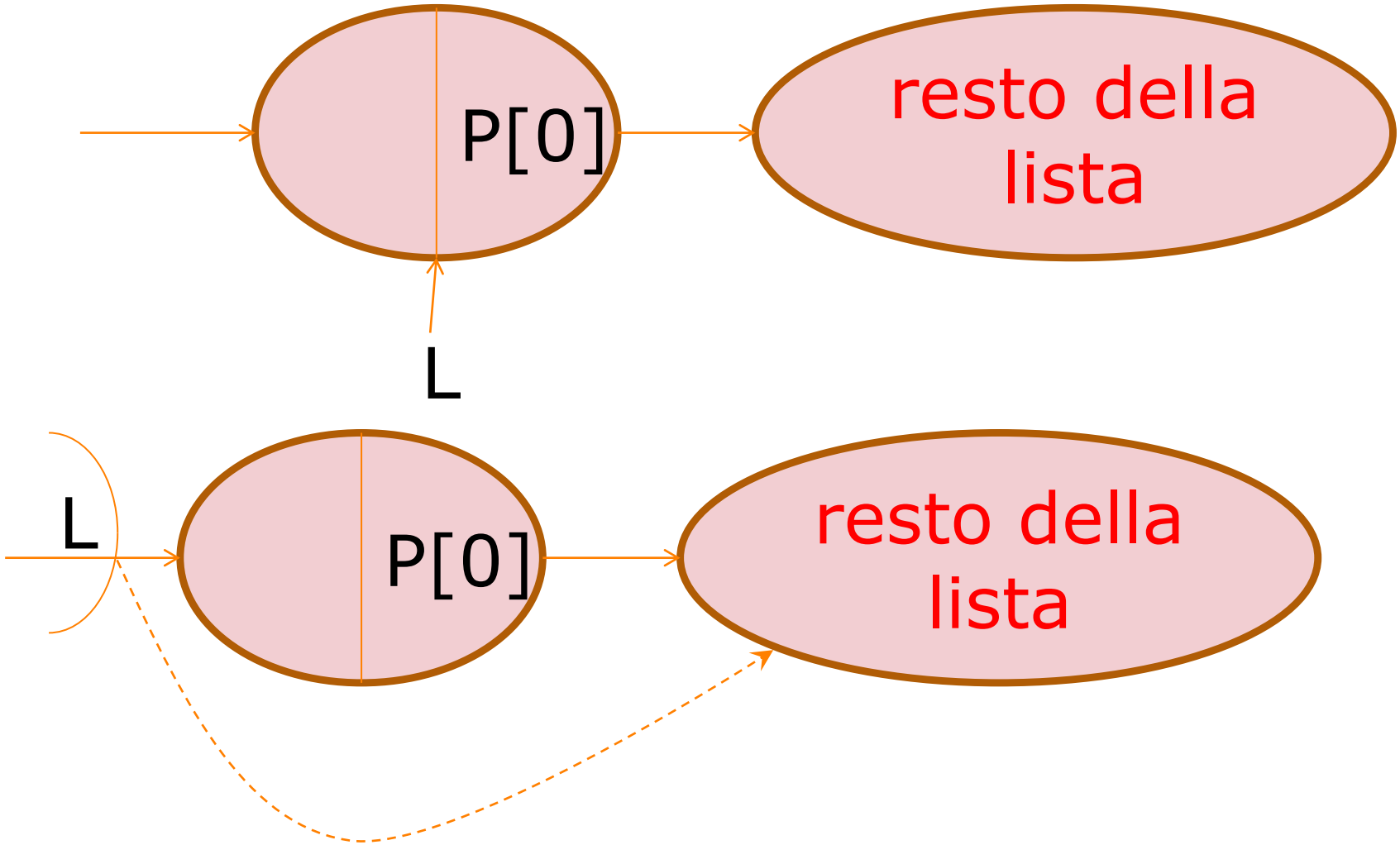
L-P significa tenere il primo nodo e collegarlo a (resto della lista)-P che è quello che restituisce

```
del(L->next, P, dimP);
```

per ipotesi induttiva

```
nodo*  del(nodo*L, char*P, int dimP)
{
   if(!L || !dimP)
     return L;
   if(L->info==P[0])
     { nodo*x=L->next; delete L;
      return del(x,P+1,dimP-1);}
   else
     {L->next=del(L->next,P,dimP);
      return L;}
}
```

soluzione con passaggio per riferimento


P[0]
resto della lista
L
L
P[0]
resto della lista

```
void  del(nodo*&L, char*P, int dimP)
{
   if(L && dimP)
   {
     if(L->info==P[0])
     { nodo*x=L; L=L->next; delete x;
      del(L,P+1,dimP-1);}
   else
      del(L->next,P,dimP);
    }
}
```