FUNZIONI e PROCEDURE

Meccanismi di astrazione

- funzioni come astrazione di operatori
- procedure " " di comandi

Funzioni : int f (- - - -) { }

Procedure : void p (- - -) { }

```
int abs (int x)
  { int ris;
    if (x >= 0) ris = x; else ris = -x;
    return ris;
  }
```

$f(x)$ calcola $|x|$

$Exp ::= abs(Exp)$

$$Sem_{exp}\ \ abs(e)\ \ \rho\ \ \mu\ \ =\ \ v$$

$$above\ \ \ v = Sem_{exp}\ \ e\ \ \rho\ \ \mu\ \ \ \ \ e\ \ \ v \geq \emptyset$$

$$Sem_{exp}\ \ abs(e)\ \ \rho\ \ \mu = -v$$

$$done\ \ \ v = Sem_{exp}\ \ e\ \ \rho\ \ \mu\ \ \ \ \ e\ \ \ v < 0$$

```
z = -w;

z = abs(w);
```

```
void swap (int *x, int *y)
  { int t;
    t = *x; *x = *y; *y = t; }
```

Com ::= Idk = Exp; | Block | ....

swap (Exp , Exp)

$$Sem_{com}\ \ swap(e1, e2)\ \rho\ \mu = \mu\left[{}^{v2}/_{l1},\ {}^{v1}/_{l2}\right]^{mod}$$

$$l1 = Sem_{exp}\ \ e1\ \rho\ \mu \qquad l1 \in Loc$$

$$l2 = Sem_{exp}\ \ e2\ \rho\ \mu \qquad l2 \in Loc$$

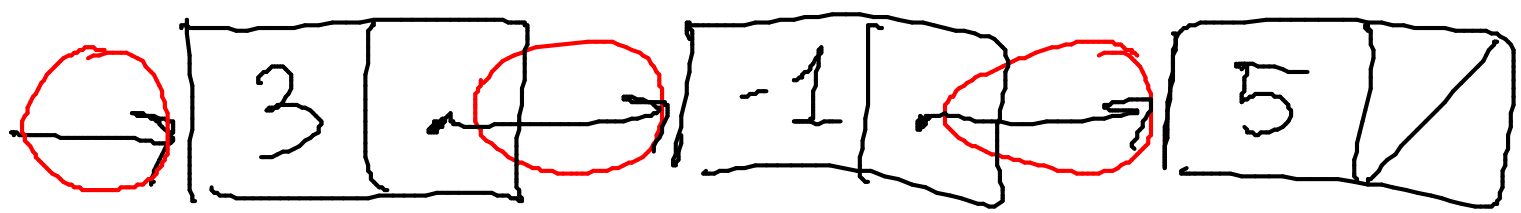
$$v1 = \mu(l1) \qquad v2 = \mu(l2)$$

```
int minmax(int a[], int dim, int *max)
/* restituisce il valore minimo presente
   nell'array a E lascia nella variabile
   puntata da max il valore massimo
   dell'array */
```

LISTE

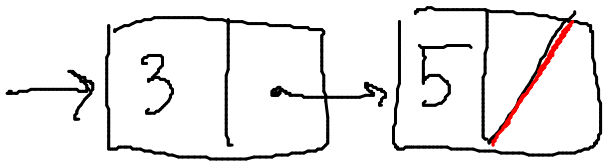
```
struct el { int info; struct el *next; }

typedef struct el Elementolista;
```



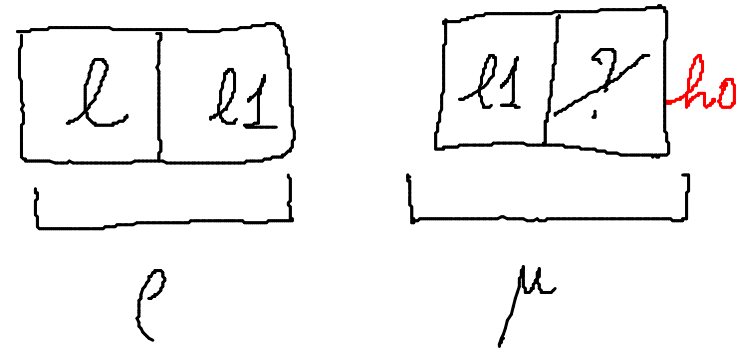

```
typedef Elementolista *listaDiElementi;
```

```
{ ListaDiElementi l;
  l = malloc (sizeof (Elementolista));
  l -> info = 3;
  l -> next = malloc (sizeof (Elementolista));
  l -> next -> info = 5;
  l -> next -> next = NULL; }
```



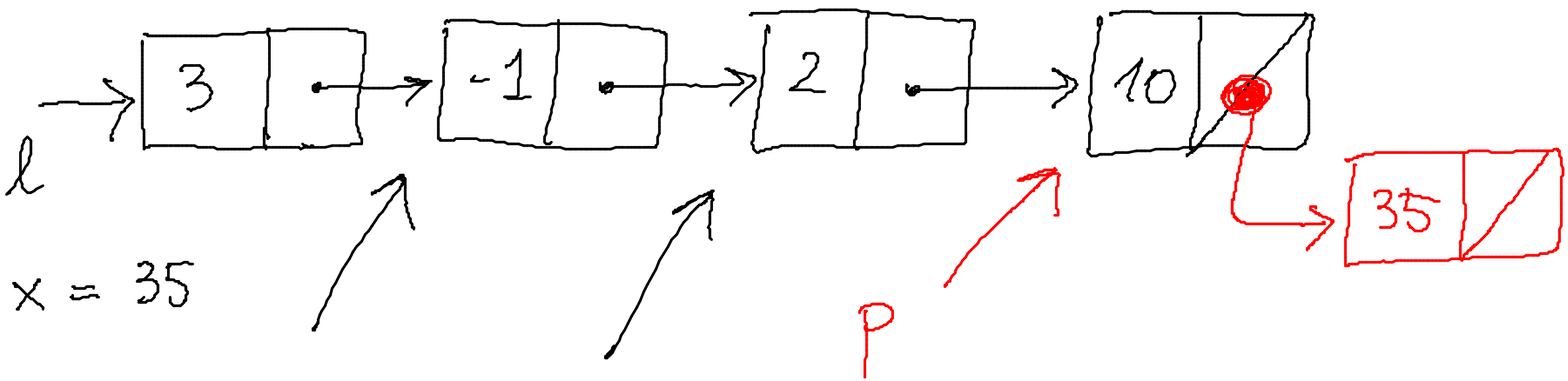
l

HEAP

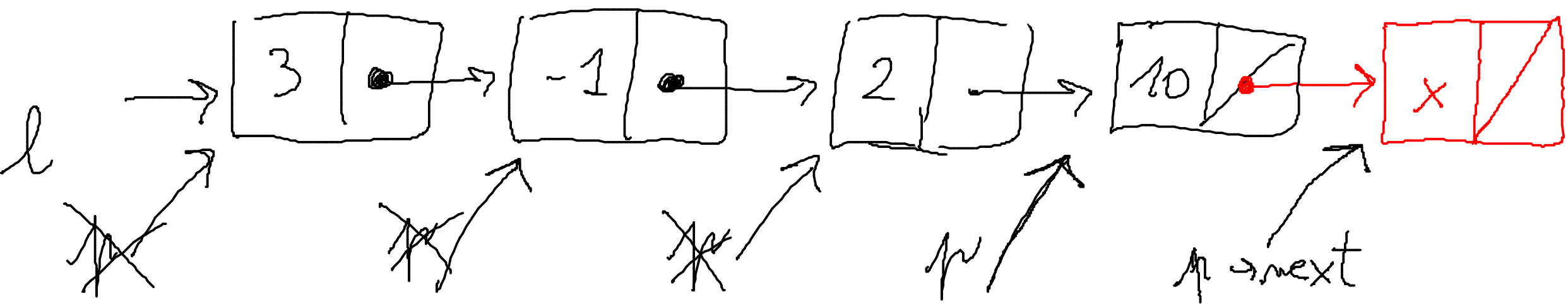


Procedura che, data una lista NON VUOTA, inserisce in coda alla lista un nuovo intero x

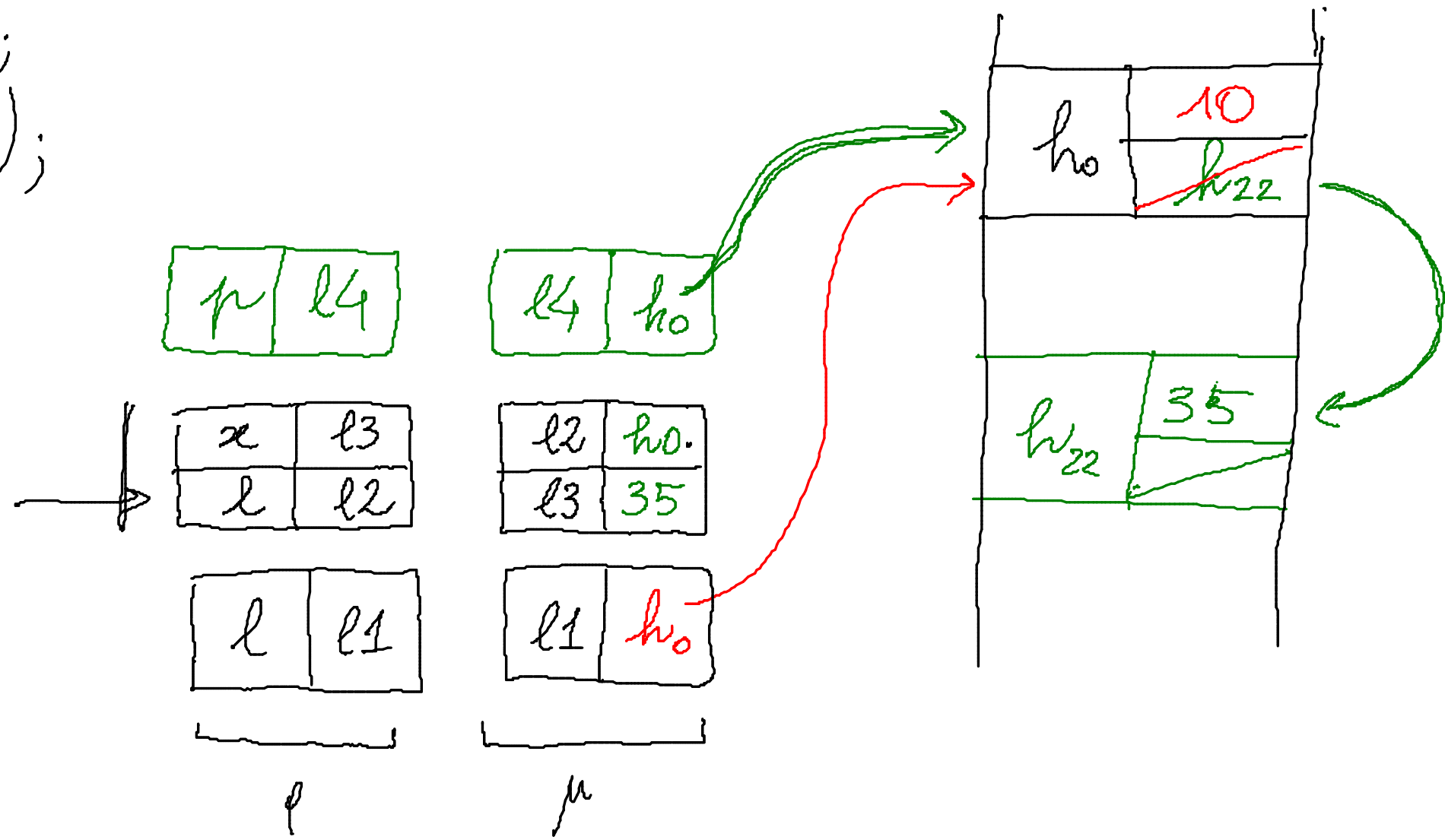
```
void InsCoda ( ListaDiElementi * l , int x)
/* l è non vuota
```



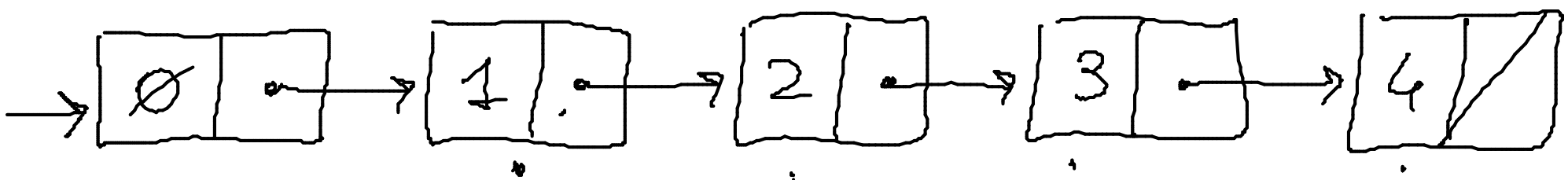
```
void InsCoda ( ListaDiElementi l , int x )
{ ListaDiElementi p = l ;
  while ( p -> next != NULL )
        p = p -> next ;
  p -> next = malloc ( sizeof ( ElementoLista )) ;
  p -> next -> info = x ;
  p -> next -> next = NULL ; }
```



```
{ ListaDiElementi l;
  l = malloc(sizeof(ElementoLista));
  l -> info = 10;
  l -> next = NULL;
  InsCoda(l, 35);
```



Costruiamo una lista fatta così




```
{ ListaDiElementi l; int i;
  l = malloc (sizeof (ElementoDiLista));
  l -> info = 0;
  l -> next = NULL;
  for (i = 1; i < 5; i++)
      InsCoda (l, i);
```

```
void InsCoda (listaD:Elemenuti l, int x)
  {  while (l->next != NULL)
        l = l->next;
     l->next = malloc( ---- );
     l->next->info = x;
     l->next->next = NULL;
  }
```

```
int member (ListaDiElementi l, int x)
{ int risultato = 0;
  while (l -> info != x)
        l = l -> next;
  if (l -> info == x) risultato = 1;
  return risultato;
}
```

NO!

ListaDiElementi l,

```
int member ( int a[], int dim, int x )
{ int trovato = 0; int i = 0;
  while ( !trovato && i < dim )
    if (a[i] == x) trovato = 1;
    else i = i+1;
  return trovato;
}
```

l != NULL

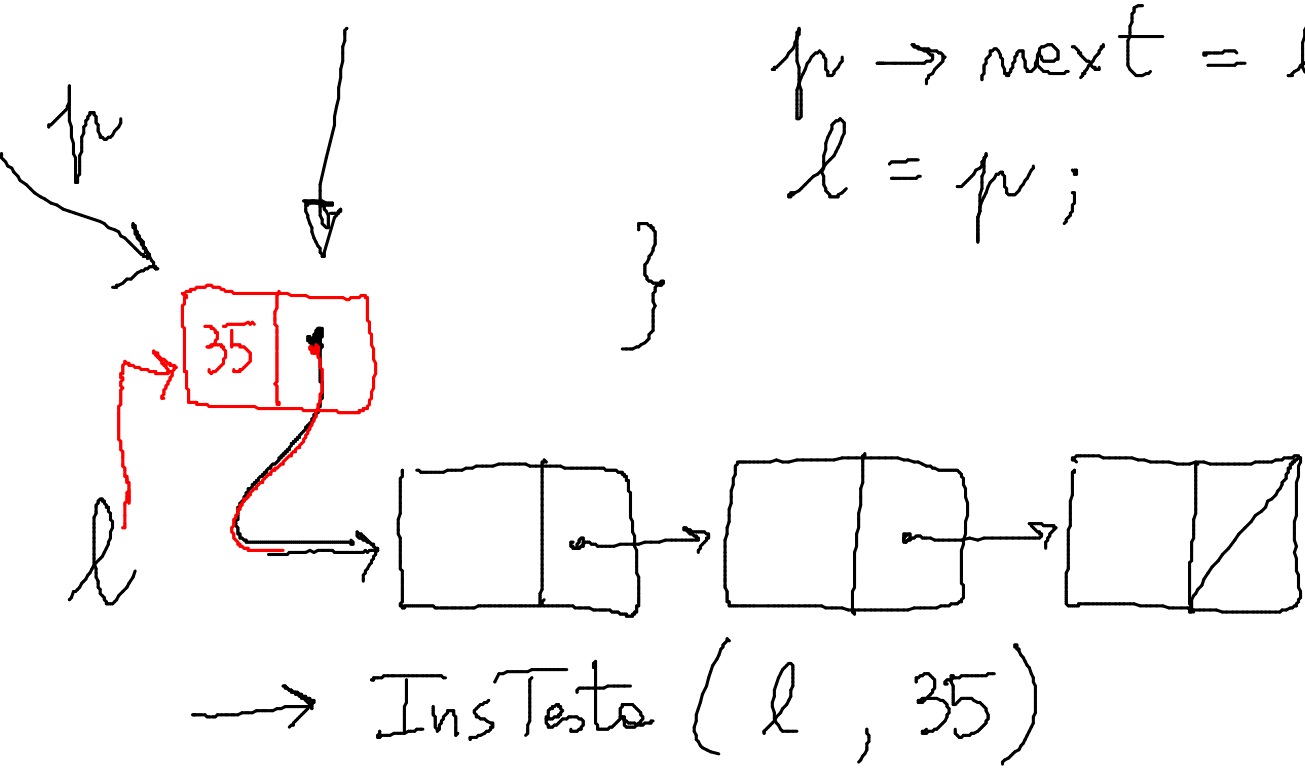
l->info == x

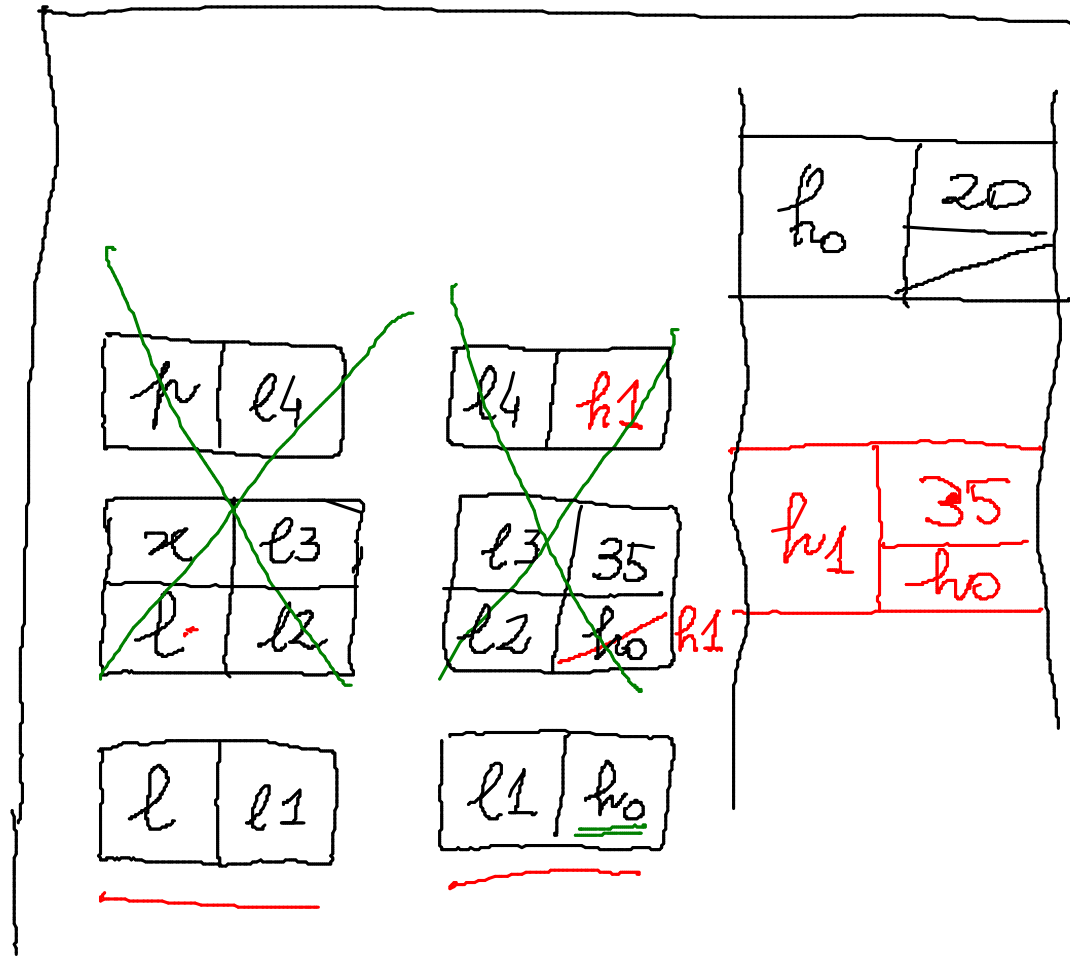
l = l->next;

```
int member ( listaDiElementi l, int x)
  { int trovato = 0;
    while ( ! trovato && l != NULL)
       if (l -> info == x) trovato = 1;
                  else  l = l -> next;
    return trovato;
  }
```

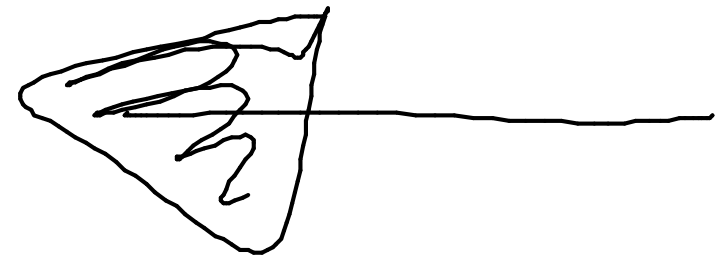
Inserire un elemento in TESTA a una lista

```
void InsTesta (ListaDiElementi l , int x)
  { ListaDiElementi p = malloc(sizeof(ElementoLista));
    p -> info = x;
    p -> next = l;
    l = p;
  }
```





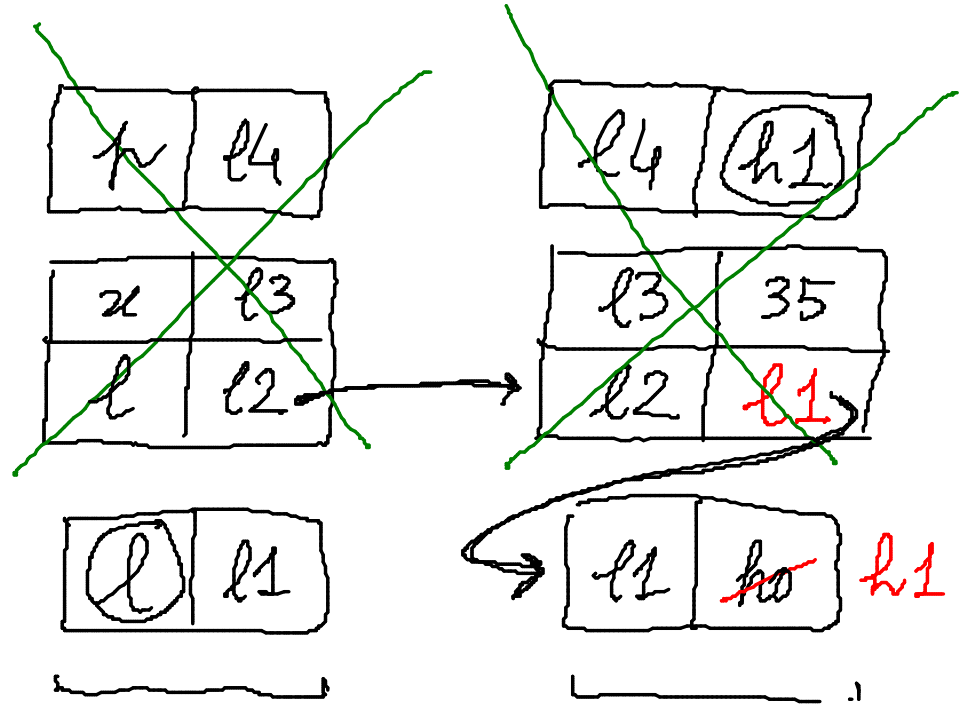
```
void incr (int *x)
    {*x = *x + 1; }
```



```
{ int y;
  incr (&y);
```

```
void InsTesta (ListaDiElementi *l, int x)
{ ListaDiElementi p = malloc(sizeof(ElementoLista));
  p -> info = x
  p -> next = *l;
  *l = p;
}
```

InsTesta(&l, 35);

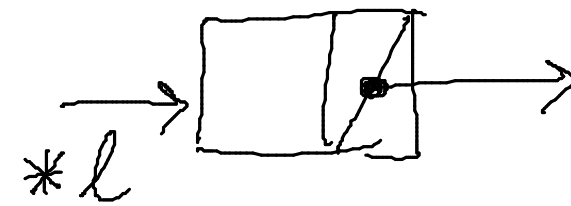


RICORDARE: tutte le volte che si scrive una procedura che (anche se in un solo caso) deve poter modificare il puntatore al primo elemento, la lista va passata per INDIRIZZO

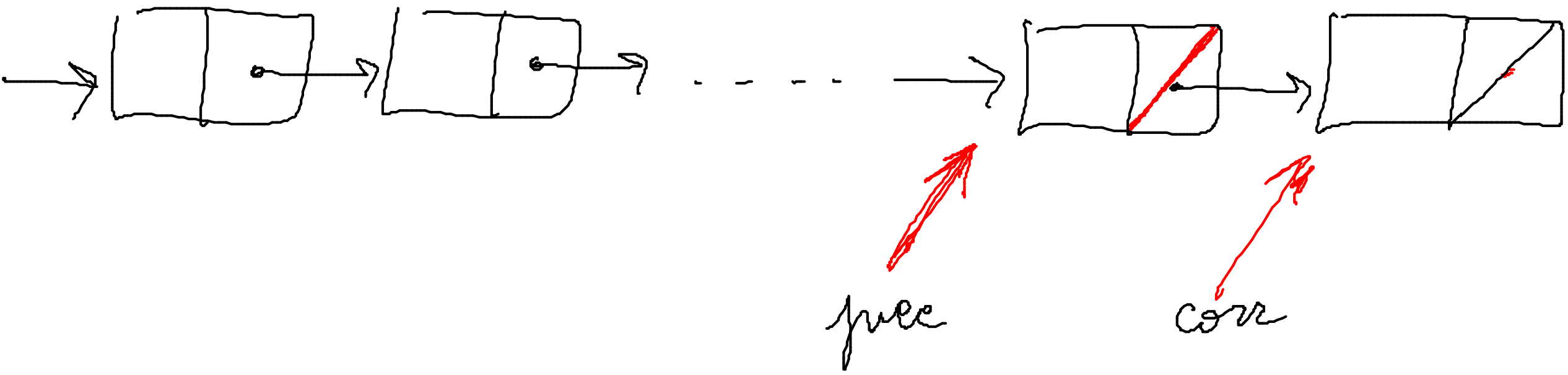
```
void InsCoda (ListaDiElementi *l, int x)
{ ListaDiElementi p = malloc(sizeof(------));
  p -> info = x;   p -> next = NULL;
  if (*l == NULL)  *l = p;
      else { ListaDiElementi scorr = *l;
             while (scorr -> next != NULL)
                 scorr = scorr -> next;
             scorr -> next = p;
           }
}
```

CANCELLARE DA UNA LISTA L'ULTIMO ELEMENTO

```
void cancultimo (ListaDiElementi *l)
  { if (*l != NULL)
      { if ((*l)->next == NULL)
          *l = NULL;
        else { ListaDiElementi corr, prec;
               prec = *l; corr = prec->next;
               while (corr->next != NULL)
                 { prec = corr;
                   corr = corr->next;
                 }
               prec->next = NULL;
             }
      }
  }
```




```
free (p);
```


free
corr