```
struct el { int info; struct el *next; }
typedef struct el ElementoDiLista;
typedef ElementoDiLista *ListaDiElementi;
```

Una lista è un oggetto di tipo ListaD.Elementi, cioè un puntatore al primo elemento della lista (o NULL se la lista è vuota.

È necessario passare una lista PER INDIRIZZO se la procedura che vogliamo definire può dover modificare il puntatore al primo elemento della lista

```
void addT ( ListaDiElementi *l, int x)
```

ListaD.Elementi * ≃ ElementoDiLista * *

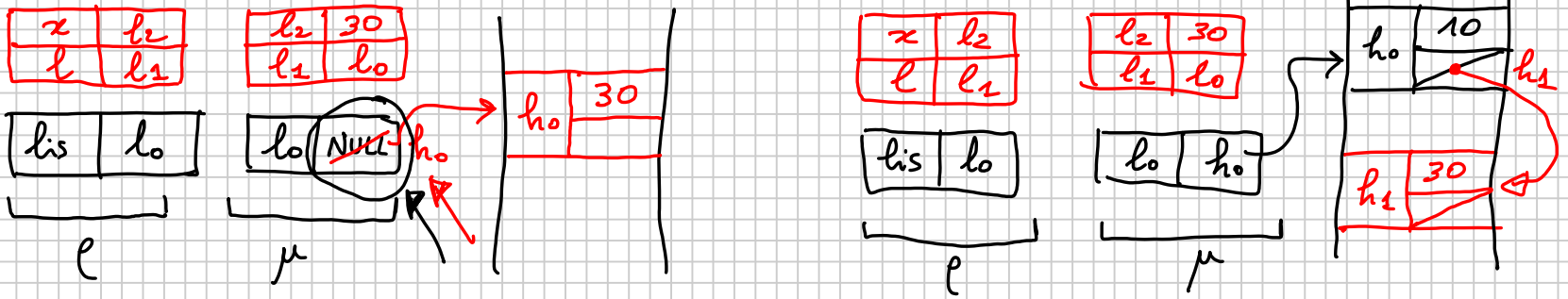
Procedura che aggiunge un nuovo elemento IN FONDO ad una lista

Bisogna individuare, attraverso un ciclo, il puntatore all'ultimo elemento della lista

Una volta individuato possiamo "agganciare" a questo elemento il nuovo elemento che contiene x, agendo sul campo next dell'elemento individuato

```
void addC ( ListaDiElementi *l , int x )
```

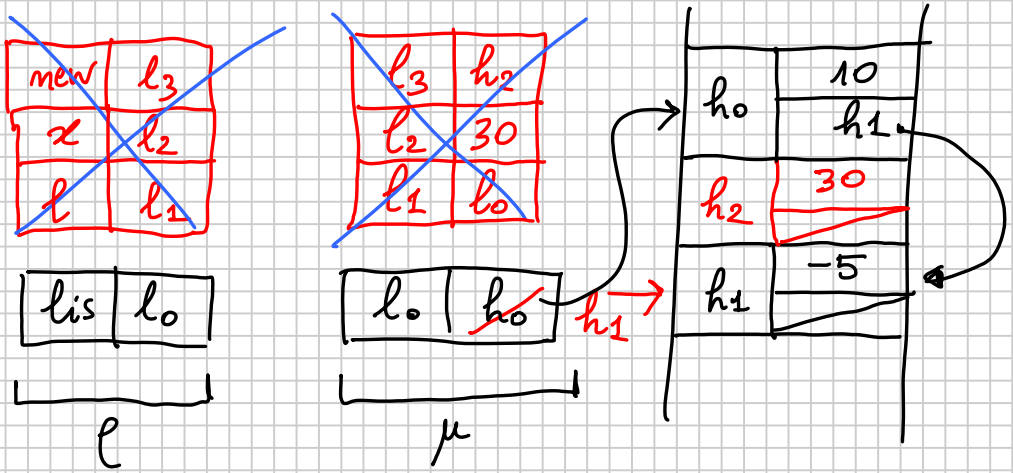
dobbiamo passare la lista per indirizzo perché, se la lista in ingresso è vuota, va modificato il valore del puntatore al PRIMO elemento

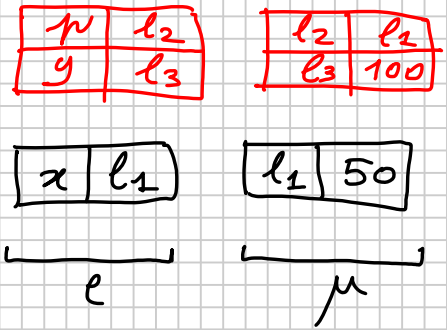


```
void addC (ListaDiElementi *l, int x)
{ ListaDiElementi new = malloc (sizeof (ElementoDiLista));
  new -> info = x;   new -> next = NULL;
  if (*l == NULL) *l = new;
  else
    while (*l -> next != NULL)
        *l = *l -> next;
}
```

No !!!
così distruggiamo la lista del chiamante

chiamata di addC(&lis, 30)



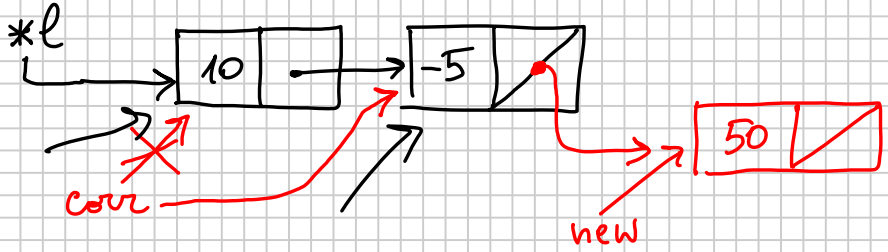



```
main ()
{ int x=50;
  foo (&x, 100);
```

```
void foo (int *p, int y)
{ *p = *p + y; }
```

```
void addC (listaDiElementi *l, int x)
{ listaDiElementi new = malloc(- - - ...);
  new -> info = x;    new -> next = NULL;
  if (*l == NULL)   *l = new;
  else
    { listaDiElementi corr = *l;
      while (corr -> next != NULL)
          corr = corr -> next;

      corr -> next = new;
    }
```

listaDiElementi corr = malloc(- - -)
corr = *l;
ok ma
crea "garbage"



addC

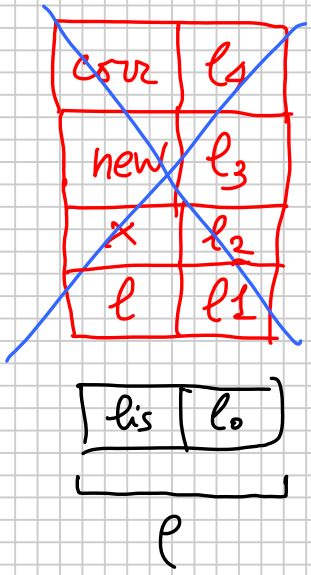
corr	l_4
l	l_1
x	l_2
new	l_3

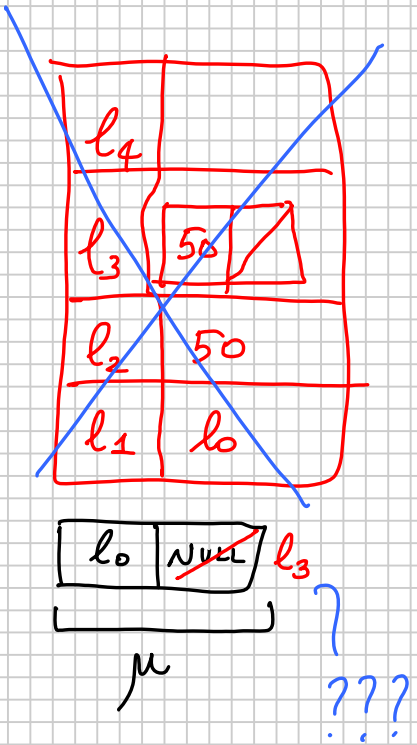
l

l_4	h_0
l_1	h_0
l_2	50
l_3	h_4

μ

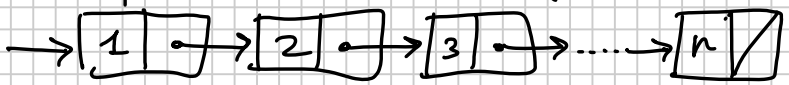
```
void addC (ListaDiElementi *l, int x)
{ ElementoDiLista new;
  new.info = x;
  new.next = NULL;
  if (*l == NULL)  *l = &new;
  else
     ⋮

  curr->next = &new;
}

addC (&lis, 50);
```



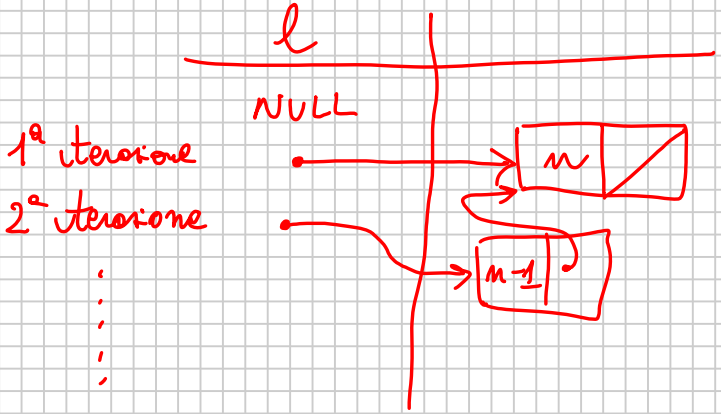


Come creare una lista che contiene i primi n numeri interi

```
#define n ...
main ()
{ ListaDiElementi l = NULL;
  .
  .
  .
/* l punta ad una lista fatta così
```




```
}
```



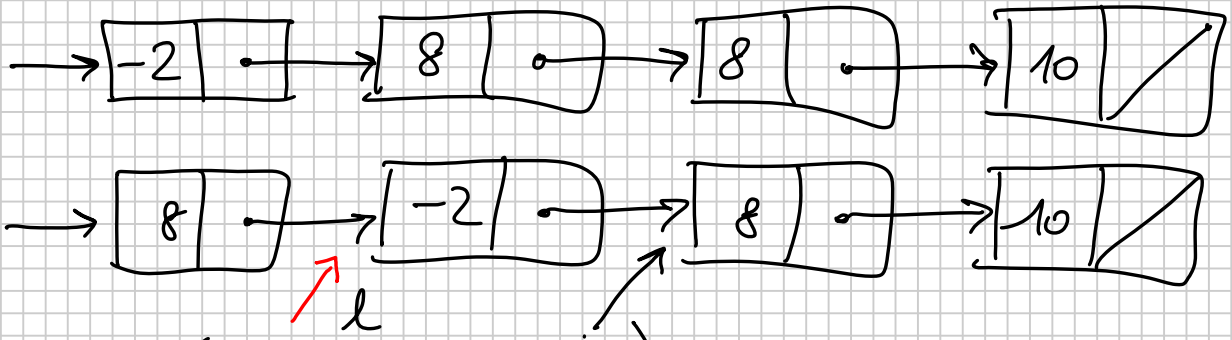

```
int i;
for (i=n; i>=1; i--)
  addT(&l, i);
```

alternativa

```
int i;
for( i=1; i<=n; i++)
  addC(&l, i);
```

meno efficiente !!!
ad ogni iterazione scandiamo tutta la lista

Data una lista, controllare se è ordinata in modo non decrescente.



✓ true

✗ false

```
int ord (ListaDiElement l)
{ int ordinata = 1;
  while ( l->next != NULL && ordinata )
    if ((l -> info) <= (l -> next -> info))
        l = l -> next;
    else ordinata = 0;
  return (ordinata);
}
```

l.info NO!!

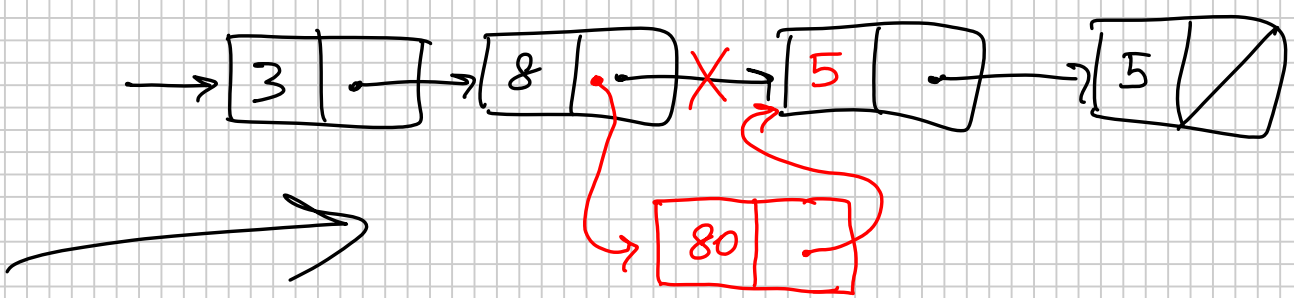
l.info si può usare se il tipo di l è struct el (ElementoDiLista)

l->info è una abbreviazione per (*l).info

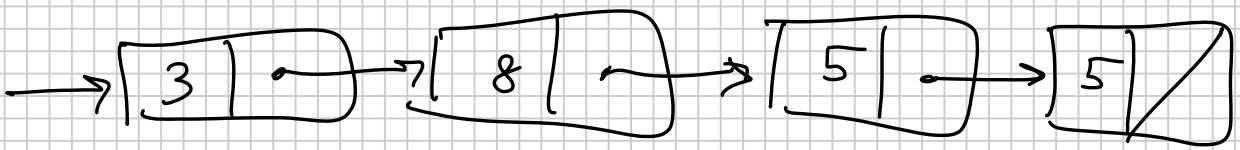
```
int ord (int a[], int dim)
{ int ordinato = 1; int i=0;
  while (i<dim-1 && ordinato)
    if (a[i] <= a[i+1]) i=i+1;
    else ordinato = 0;
  return (ordinato);
}
```

Stesso problema!

Scrivere una procedura che inserisce in una lista un valore V prima della prima occorrenza di un dato valore x. Se x non occorre nella lista, questa deve rimanere invariata.



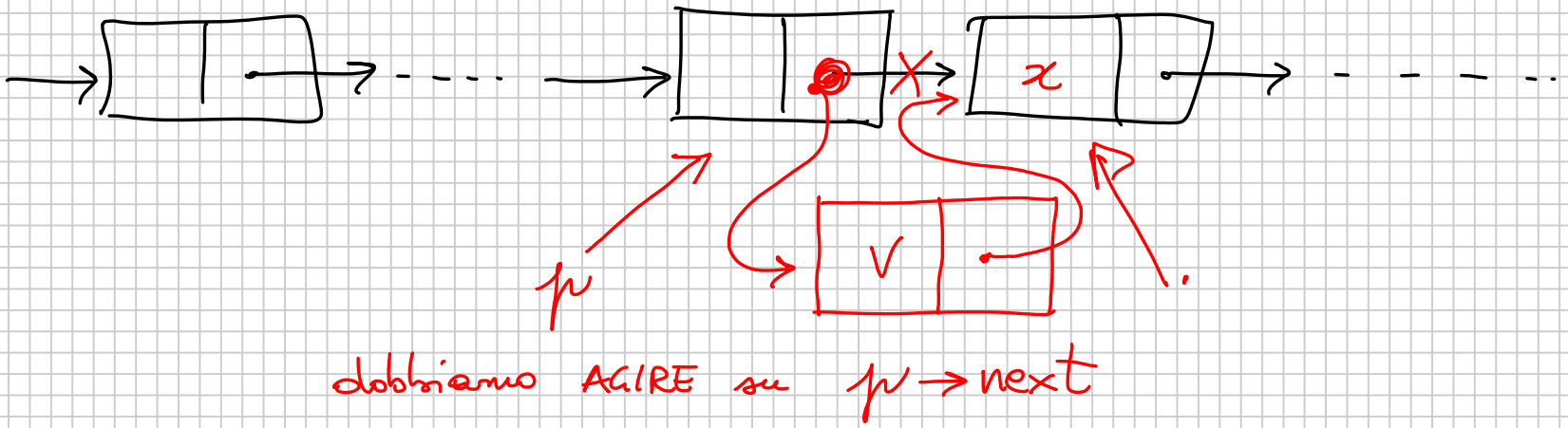
V = 80
x = 5



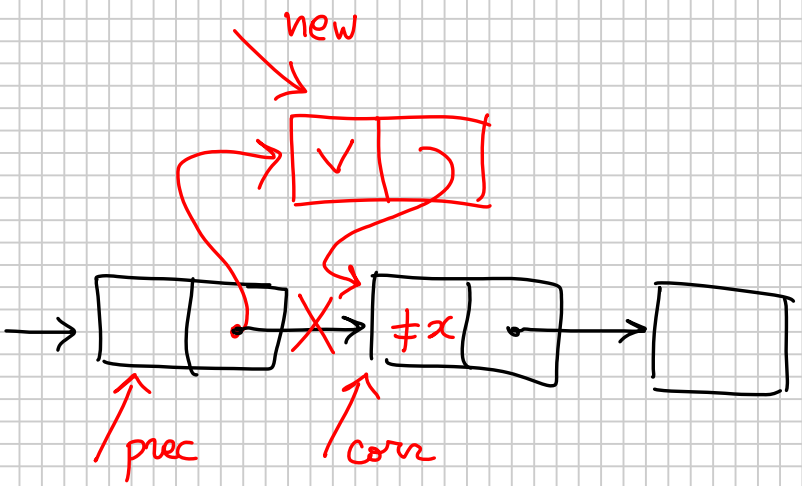
V = 80
x = 100

Void insprima (ListaDiElementi *l, int x, int v)

va passata per indirizzo perché x potrebbe occorrere in prima posizione.


x
v
p
dobbiamo AGIRE su p → next

```
void insprima (ListaDiElementi *l, int x, int v)
{ ListaDiElementi prec, corr;
  prec = NULL; corr = *l;
  int trovato = 0;
  while (corr != NULL && !trovato)
     if (corr -> info == x) trovato = 1;
     else
          { prec = corr;
            corr = corr -> next;
          }
  if (trovato)
    { ListaDiElementi new = malloc(....);
      new -> info = v; new -> next = corr;
      if (prec != NULL) prec -> next = new; else *l = new;
    }
}
```



Variante: inserire x in fondo alla lista se x non occorre nella lista.

per esercizio