OBIETTIVO LEGGIBILITA': precondizioni e postcondizioni, ma anche NOMI AUTOESPLICATIVI.

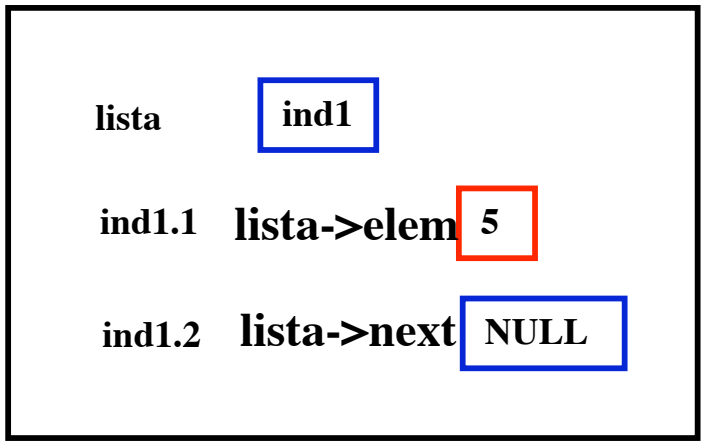
```
 void insListOrd (ListaPtr *L, int val)
{/*prec: La lista deve essere ordinata in ordine crescente*/
 /*postc: inserisce val nella prima posizione consistente con l'ordine */
ListaPtr new;
if (*L == NULL || (*L)->elem >= val)
    {new = malloc(sizeof(Nodo));
     new->elem = val;
     new->next = *L;
     *L = new;
     }
   else insListOrd(&(*L)->next, val);
}
```

Un programma si scrive una volta, ma si legge e si usa molte volte!

```
 void insTestaListSb (ListaPtr L, int val)
{ListaPtr new;
   {new = malloc(sizeof(Lista));
    new->elem = val;
    new->next = L;
    L = new;
    printf(" dopo l'inserimento, ma nella chiamata \n");
stampaListaInt(L);}
}
main()
{/* facciamo vedere perchè insTestaListSb è un'implementazione
sbagliata */
int es = 5, es1 = 8;
ListaPtr lista;
lista = malloc(sizeof(lista));
lista->elem = es;
Lista->next = NULL;
printf(" prima di un qualsiasi inserimento \n");
stampaListaInt(lista);
insTestaListSb(lista, es);
printf(" dopo l'inserimento \n");
stampaListaInt(lista);
return 0;}
```

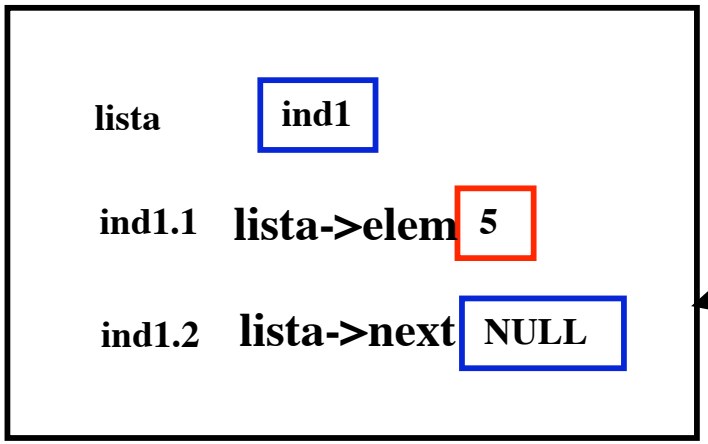
L'effetto delle seguenti istruzioni in memoria è:

```
lista = malloc(sizeof(lista));
lista->elem = es;
lista->next = NULL;
printf(" prima di un qualsiasi inserimento \n");
stampaListaInt(lista);
```



OUTPUT:

```
prima di un qualsiasi inserimento
la lista  é
5 --> NULL
```

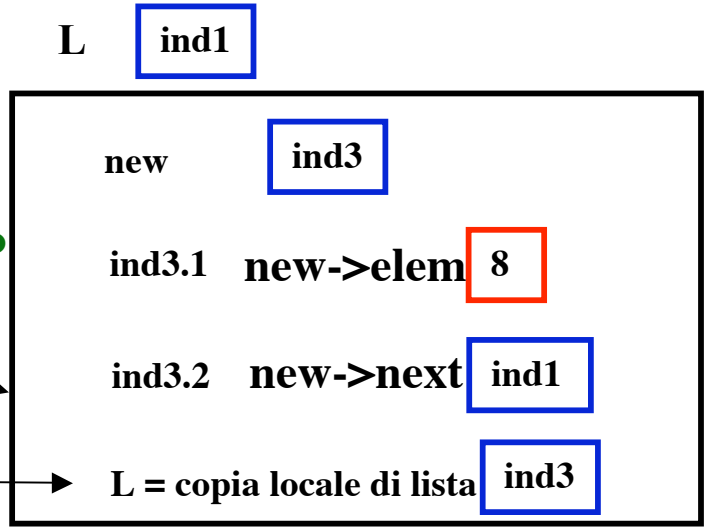


Situazione memoria prima della chiamata

La chiamata
insTestaListsb(lista, es);
produce la copia del valore di lista
nel parametro formale L

E l'esecuzione del seguente codice produce l'effetto

```
new = malloc(sizeof(L));
new->elem = val;
new->next = L;
L = new;
```



dopo l'inserimento, ma nella chiamata
la lista é
8 --> 5 --> NULL

Ma all'uscita dalla chiamata la modifica su L è persa!

dopo l'inserimento, fuori chiamata
la lista é
5 --> NULL

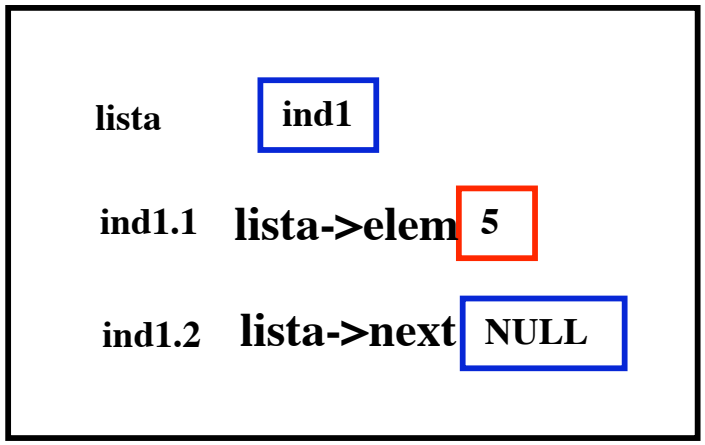
Vediamo un esempio in cui esportiamo le modifiche in memoria.

```
void insSecList(ListaPtr L, int val)
{ListaPtr new;
   {new = malloc(sizeof(Lista));
    new->elem = val;
    new->next = NULL;
    L ->next = new;
    printf(" dopo l'inserimento, ma nella chiamata \n");
stampaListaInt(L);}
}
```

```
main()
{/* facciamo vedere perchè insSecList funziona */
int es = 5, es1 = 8;
ListaPtr lista;
lista = malloc(sizeof(lista));
lista->elem = es;
lista>next = NULL;
printf(" prima di un qualsiasi inserimento \n");
stampaListaInt(lista);
insSecList(lista, es);
printf(" dopo l'inserimento \n");
stampaListaInt(lista);
return 0;}
```

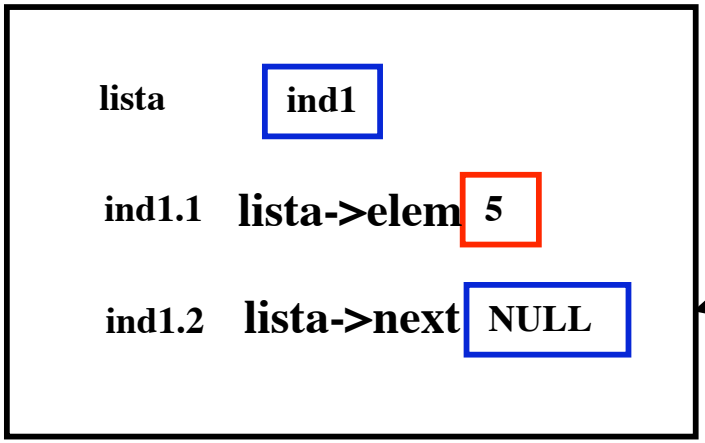
L'effetto delle seguenti istruzioni in memoria è:

```
lista = malloc(sizeof(lista));
lista->elem = es;
lista->next = NULL;
printf(" prima di un qualsiasi inserimento \n");
stampaListaInt(lista);
```



OUTPUT:

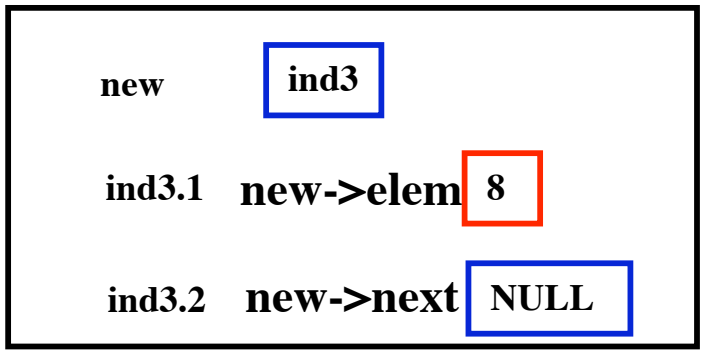
```
prima di un qualsiasi inserimento
la lista  é
5 --> NULL
```



Situazione memoria prima della chiamata

La chiamata
insSecList(lista, es);
produce la copia del valore di lista
nel parametro formale L

L = copia locale di lista ind1



E l'esecuzione del seguente codice produce l'effetto

```
new = malloc(sizeof(L));
new->elem = val;
new->next = NULL;
L ->next = new;
```

ind1.2 ind3

dopo l'inserimento, nella chiamata
(ma anche fuori chiamata) la lista é
5 --> 8 --> NULL

Quindi lista->next ora contiene l'indirizzo di new!!

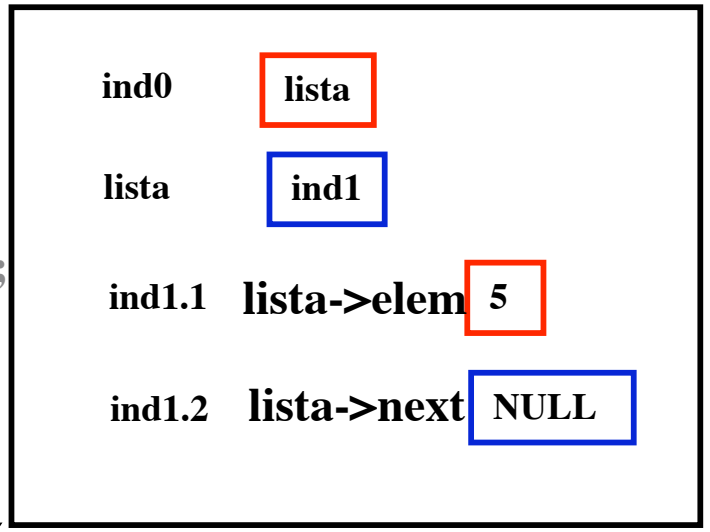
L'inserimento ricorsivo in una lista ordinata, avviene sempre come se si inserisse in testa a una lista, perché in ogni chiamata il valore del parametro è il puntatore al successivo elemento della lista. Quindi se si scrive:

```
void insListOrdSb (ListaPtr L, int val)
{ListaPtr new;
  if (L == NULL || L ->elem >= val)
  {new = malloc(sizeof(Lista));
     new->elem = val;
     new->next = L;
     L = new;
     printf(" dopo l'inserimento, ma nella chiamata \n");
     stampaListaInt(L);}
else insListOrdSb(L->next, val);
printf("all'uscita dalla chiamata \n");
stampaListaInt(L);}
```

NON si ottiene affatto l'effetto voluto, anzi eseguendo la funzione sempre sugli stessi dati di ingresso si otterrebbe:

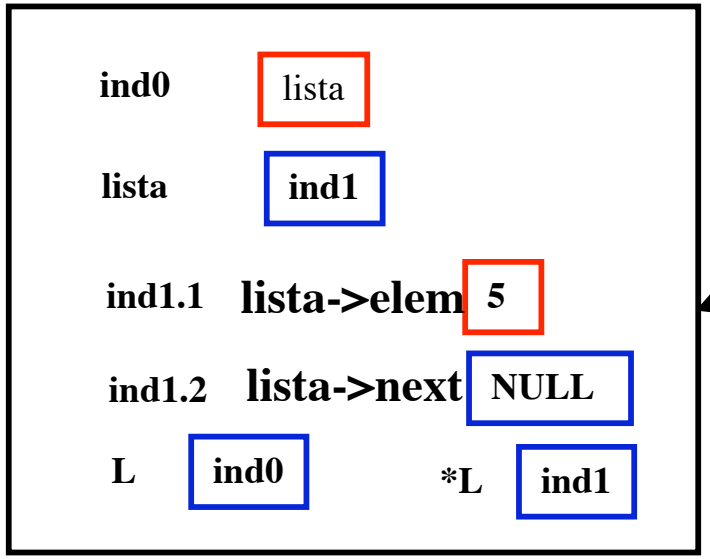
```
prima di un qualsiasi inserimento
la lista  é
5 --> NULL
dopo l'inserimento, ma nella chiamata
la lista  é
8 --> NULL
all'uscita dalla chiamata
la lista  é
8 --> NULL
all'uscita dalla chiamata
la lista  é
5 --> NULL
dopo l'inserimento, fuori chiamata
la lista  é
5 --> NULL
```

```
main()
{/* facciamo vedere perchè insListOrd funziona */
int es = 5, es1 = 8;
ListaPtr lista;
lista = malloc(sizeof(lista));
lista->elem = es;
lista>next = NULL;
printf(" prima di un qualsiasi inserimento \n");
stampaListaInt(lista);
insListOrd(&lista, es1);
printf(" dopo l'inserimento \n");
stampaListaInt(lista);
return 0;}
```



Situazione memoria prima della chiamata

La chiamata
insListOrd(&lista, es);
produce in L la copia del valore dell'indirizzo di lista



Situazione memoria prima della chiamata

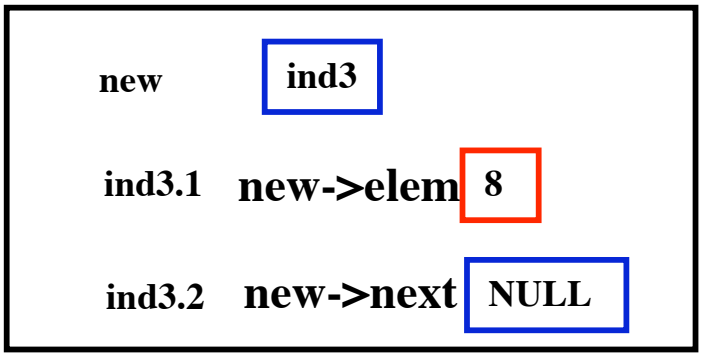
La prima chiamata produce solo una seconda chiamata:

insert (&(*L)->next, val);

il cui effetto è quello di inserire nel parametro formale L (copia locale alla chiamata) l'indirizzo del puntatore al prossimo elemento nella lista.
Poichè *L(IIc) è NULL si esegue il seguente codice

L(IIc) **ind1.2** ***L(IIc)** **NULL**

```
new = malloc(sizeof(Nodo));
new->elem = val;
new->next = *L;
*L = new;
```



***L(IIc)** **ind3**

quindi ind1.2 contiene ind3 e non più NULL!