Inserimento di un elemento in una lista

- In testa a una lista ✅

  facile da realizzare

  l'ordine in lista è esattamente inverso rispetto all'ordine di inserimento

- In fondo a una lista ✅

  facile da realizzare?

  l'ordine in lista è esattamente identico all'ordine di inserimento

- Inserimento ordinato

  facile da realizzare?

  l'ordine in lista rispetta la relazione d'ordine tra elementi utilizzata (crescente / decrescente / etc)

Inserimento in testa (*recap*)

```
#include <stdio.h>
#include <stdlib.h>
typedef struct list_element { int value;
              struct list_element *next; } item;
typedef item *list;
void showList(list l);

list cons(int e, list l) { list t;
       t = (list) malloc(sizeof(item));
       t->value = e; t->next = l;
       return t; }

main() {
list root = NULL; int i;
do {   printf("\nIntrodurre valore: \t");
       scanf("%d", &i);
       root = cons(i, root);
  } while (i!=0);
showList(root); }
```

Inserimento in fondo (*recap*)

```
// FUNZIONE CHE INSERISCE IN FONDO - PRIMITIVA ITERATIVA

list  cons_tail(int e, list l) {
  list prec, aux;
  list patt=l;

  aux=(list)malloc(sizeof(item));          // ALLOCA NODO
  aux->value=e;
  aux->next=NULL;
  if (l==NULL) return aux;     // INSERISCE IN LISTA VUOTA
  else
    { while (patt!=NULL)              // NON FINE LISTA
        { prec=patt ;
          patt=patt->next; }

      prec->next=aux;                 // AGGIUNGE IN FONDO
      return l;                       // RESTITUISCE RADICE l
    }
}
```

LISTE ORDINATE

Deve essere definita una ***relazione d'ordine*** sul ***dominio-base*** degli elementi della lista

NOTA: criterio di ordinamento dipende da ***dominio base*** e dalla specifica ***necessità applicativa***

Ad esempio:

- interi ordinati in senso crescente, decrescente, …
- stringhe ordinate in ordine alfabetico, in base alla loro lunghezza, …
- persone ordinate in base all'ordinamento alfabetico del loro cognome, all'età, al codice fiscale, …

•Vedremo tre versioni della funzione …

Inserimento ordinato in lista

- Iterativa (vers. it)

  Scandisce la lista con due puntatori ausiliari, fino al punto in cui va inserito l'elemento, e aggiunge un nodo sistemando i puntatori

- Ricorsiva
  - Senza duplicazione (no structure copying), vers. r1
  - Con duplicazione di parte della lista (con structure copying), vers. r2

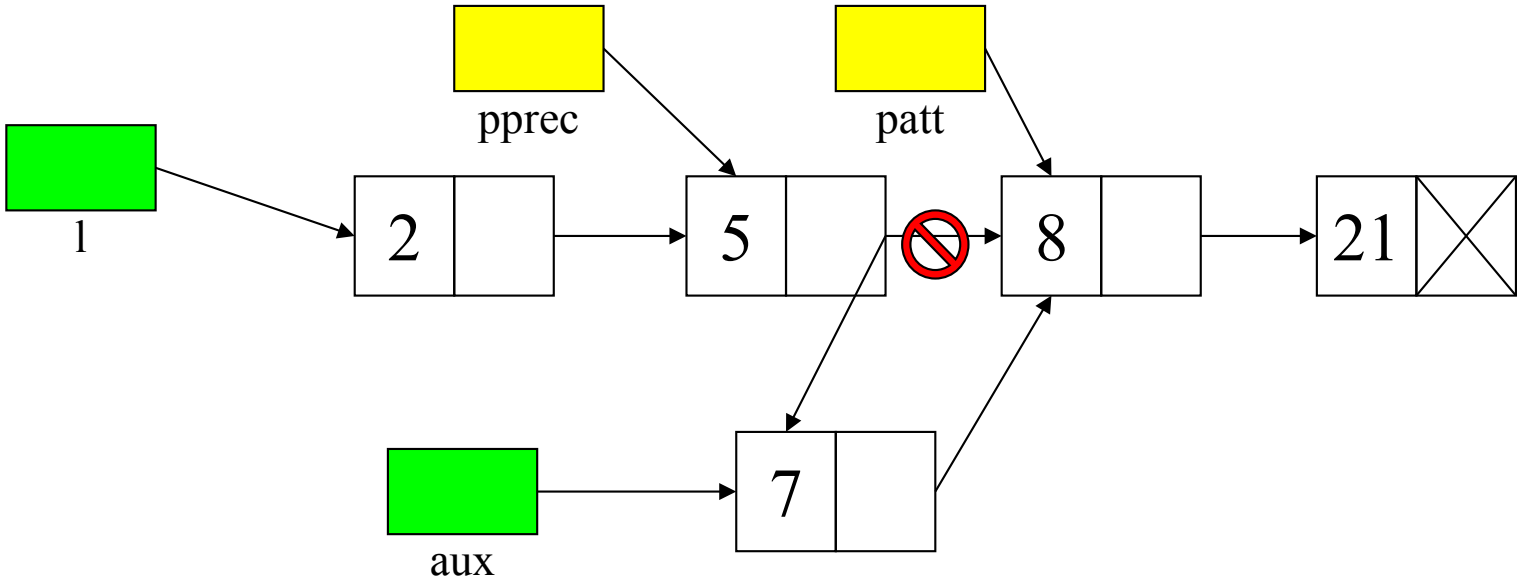
La funzione insord iterativa (vers. it)

Versione primitiva e ***iterativa*** della funzione ***insord***, sapendo che la lista di partenza è ordinata

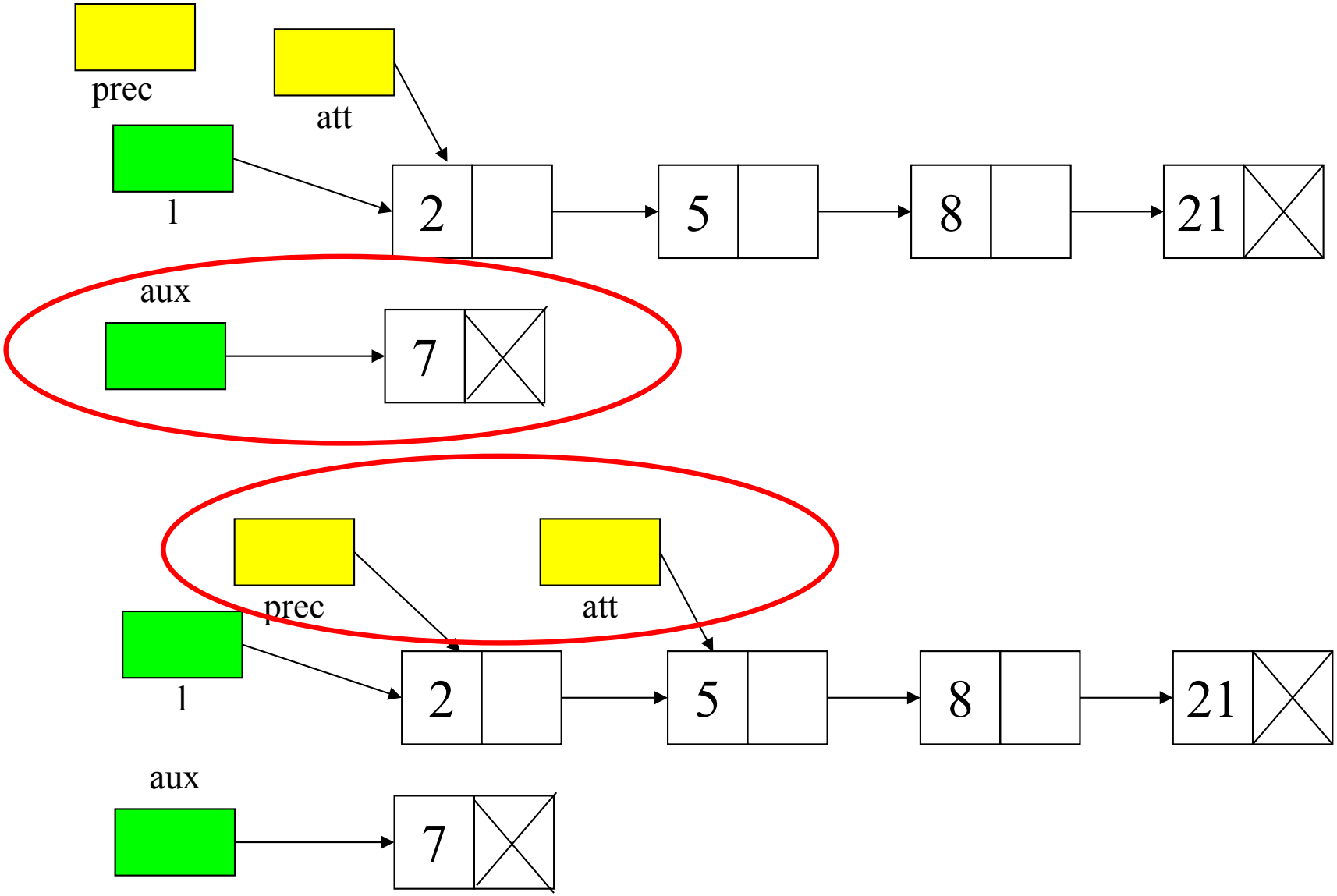
Algoritmo:

- Scandire la lista finché si incontra un nodo contenente un elemento maggiore di quello da inserire
- Allocare un nuovo nodo, con l'elemento da inserire
- Collegare il nuovo nodo ai due adiacenti (vedi figura)

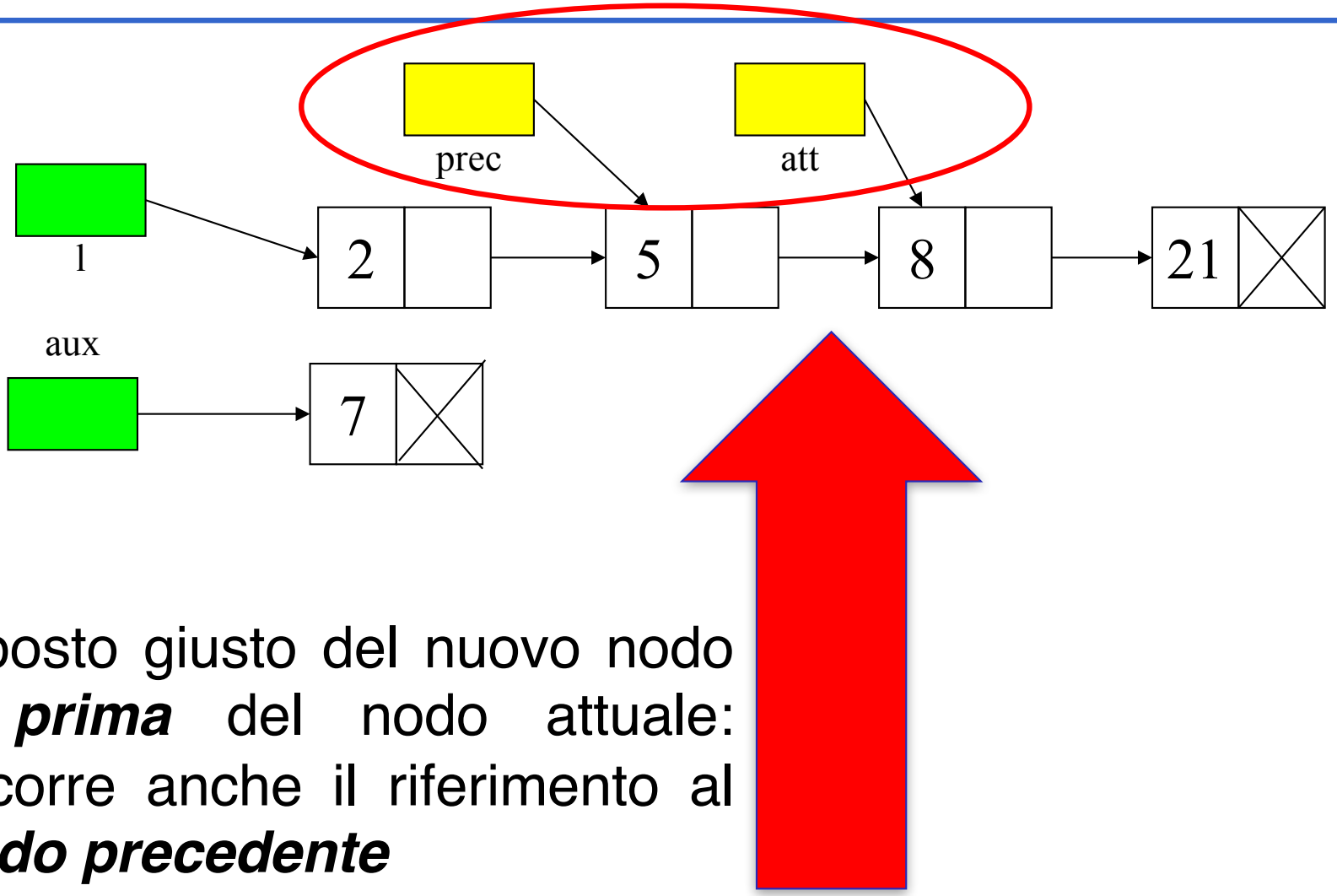
Somiglia all'inserimento in fondo visto?



Inserimento ordinato iterativo



Inserimento in fondo a una lista



Il posto giusto del nuovo nodo è ***prima*** del nodo attuale: occorre anche il riferimento al ***nodo precedente***

Inserimento in fondo

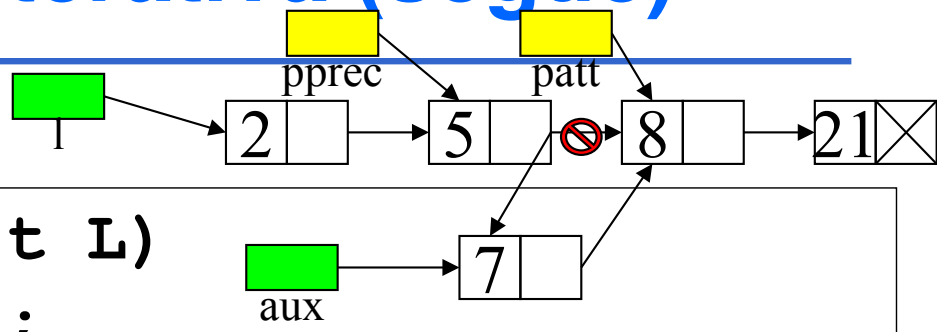
```
// FUNZIONE CHE INSERISCE IN FONDO - PRIMITIVA ITERATIVA

list  cons_tail(int e, list l) {
  list prec, aux;
  list patt=l;

  aux=(list)malloc(sizeof(item));          // ALLOCA NODO
  aux->value=e;
  aux->next=NULL;
  if (l==NULL) return aux;     // INSERISCE IN LISTA VUOTA
  else
    { while (patt!=NULL)              // NON FINE LISTA
        { prec=patt ;
          patt=patt->next; }

      prec->next=aux;                 // AGGIUNGE IN FONDO
      return l;                       // RESTITUISCE RADICE l
    }
}
```

La funzione insord iterativa (segue)




```
list insord_p(int el, list L)
{ list pprec, patt=L, aux;
  int trovato=0;
  while (patt!=NULL && !trovato)
  { if (el <= patt->value) trovato = 1;
    else {pprec=patt; patt=patt->next; }  }

  aux=(list) malloc(sizeof(item));
  aux->value = el;
  aux->next = patt;
  if (patt==L) return aux;    //inizio lista
  else { pprec->next = aux; return L
; }
}
```

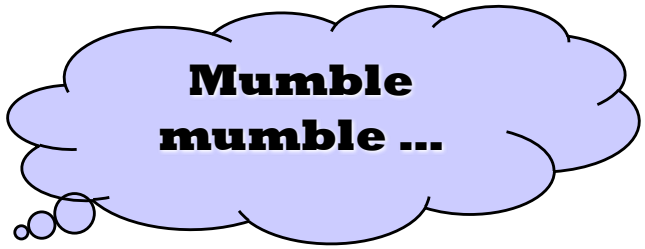
Creazione di tre liste

Riscriviamo il ***main*** creando tre liste, L1, L2, e L3, usando ***cons***, ***cons_tail*** e ***insord_p*** per inserire l'elemento letto:

```
L1= cons( i, L1);
L2= cons_tail( i, L2);
L3= insord_p( i, L3);
```

e stampiamole poi con tre chiamate a:

```
showList( );
```



Leggiamo un intero e cerchiamolo nelle tre liste:

```
member(i, L );
```

Per la lista ordinata L3, cambia la complessità della **`member`**?

ESERCIZIO : ricerca in una lista

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

È sempre una ***scansione sequenziale,*** **anche se la lista è ordinata !**

- nel caso ***peggiore***, occorre scandire l'intera lista *O(N)*
- nel caso ***medio***, proporzionale a N/2 *O(N)*

Nota Bene: L'accesso agli elementi della lista è sempre sequenziale (dal primo all'ultimo), non c'è accesso diretto a elementi intermedi!

RICORSIONE: `showListr` ricorsiva

La lista va ***scandita (sequenzialmente) con un puntatore,*** *oppure* ***versioni ricorsive***

Stampa di una lista (ricorsiva):

```
void showListr(list l) {
      if( l!=NULL )
            { printf("%d", l->value);
              showListr( l->next ); }
}
```

ESERCIZIO 3bis: ricerca in una lista

Soluzione ricorsiva:

```
int member_r(int e, list l) {
      if (l!=NULL)
            if (l->value == e) return 1;
            else return member_r(el,l->next);
      return 0;
}
```

LISTE ORDINATE: la funzione insord

Versione iterativa (vers. it), con puntatori pprec e patt con cui si scandisce la lista, quella appena vista

E' possibile realizzare una versione ricorsiva?

- Se la lista è vuota … aggiungo un nodo in testa
- Altrimenti, se l'elemento da inserire è più piccolo di quello in testa alla lista …aggiungo un nodo in testa
- Altrimenti, aggiungo l'elemento nel resto della lista e …

LISTE ORDINATE: la funzione insord

Versione iterativa (vers. it), con puntatori pprec e patt con cui si scandisce la lista, quella appena vista

E' possibile realizzare una versione ricorsiva?

- Se la lista è vuota … aggiungo un nodo in testa
- Altrimenti, se l'elemento da inserire è più piccolo di quello in testa alla lista …aggiungo un nodo in testa
- Altrimenti, aggiungo l'elemento nel resto della lista e …
  - Versione ricorsiva r1
  - Versione ricorsiva r2

LISTE ORDINATE: la funzione insord (r1)

Per inserire un elemento in modo ordinato in una lista supposta ordinata:

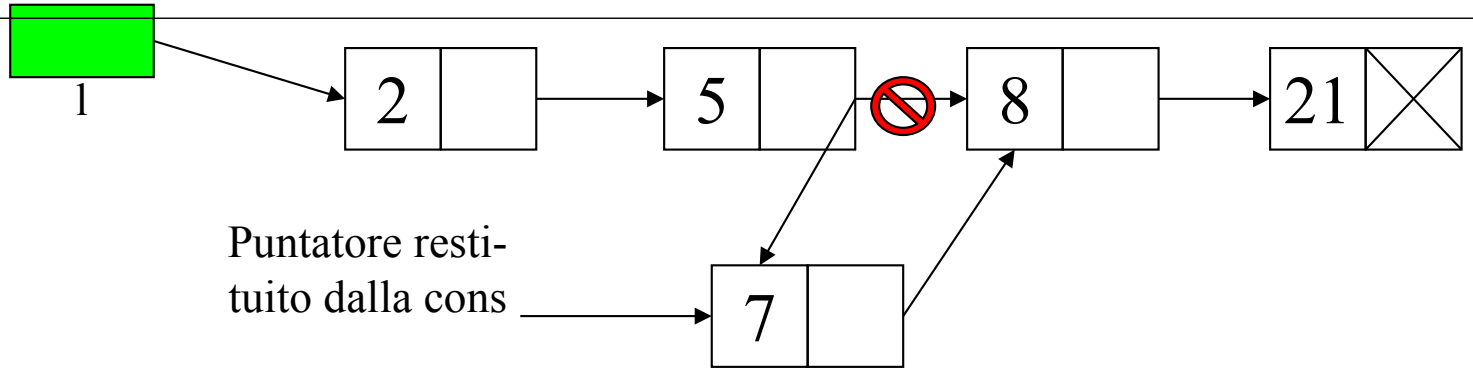
- se la lista è vuota, costruire una ***nuova lista*** contenente il nuovo elemento, *altrimenti*
- se l'elemento da inserire è minore della testa della lista, aggiungere il ***nuovo elemento in testa*** alla lista data, *altrimenti*
- l'elemento andrà ***inserito più avanti*** della lista data

I primi due casi sono operazioni elementari.
Il terzo caso può essere trattato ***in modo iterativo*** o riconducendosi allo ***stesso problema ricorsivamente, ma su un caso più semplice***:

La funzione insord ricorsiva (segue)

Il terzo caso può essere trattato riconducendosi allo ***stesso problema ricorsivamente, ma su un caso più semplice***: alla fine si potrà effettuare o un inserimento in testa o ci si ricondurrà alla lista vuota

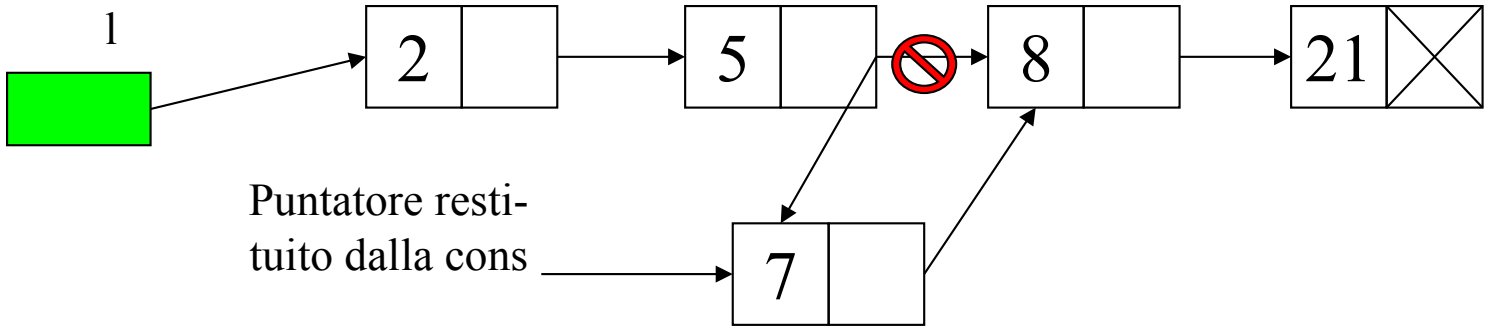
```
list insord(int el, list l) {
  if (l==NULL) return cons(el, l);
  else if (el<=l->value) return cons(el, l);
     else <inserisci el nel resto di l, e
           restituisci la lista così modificata>
}
```



La funzione insord ricorsiva (vers.r1)

Il terzo caso può essere trattato riconducendosi allo ***stesso problema ricorsivamente, ma su un caso più semplice***: alla fine si potrà effettuare o un inserimento in testa o ci si ricondurrà alla lista vuota

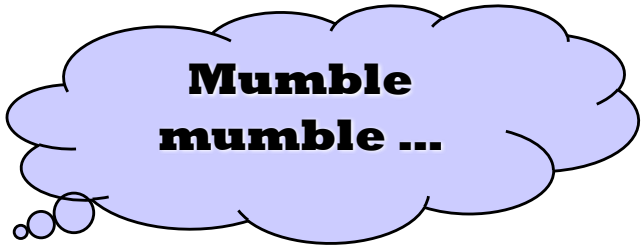
```
list insord(int el, list l) {
  if (l==NULL) return cons(el, l);
  else if (el<=l->value) return cons(el, l);
     else {l->next = insord(el,l->next);
           return l; }
}
```



DO IT!

Riscriviamo il ***main*** creando tre liste, L1, L2, e L3, usando ***cons***, ***cons_tail*** e ***insord (insord_p)*** per inserire l'elemento letto:

```
L1= cons( i, L1);
L2= cons_tail( i, L2);
L3= insord( i, L3);
```



e stampiamole poi con tre chiamate a:

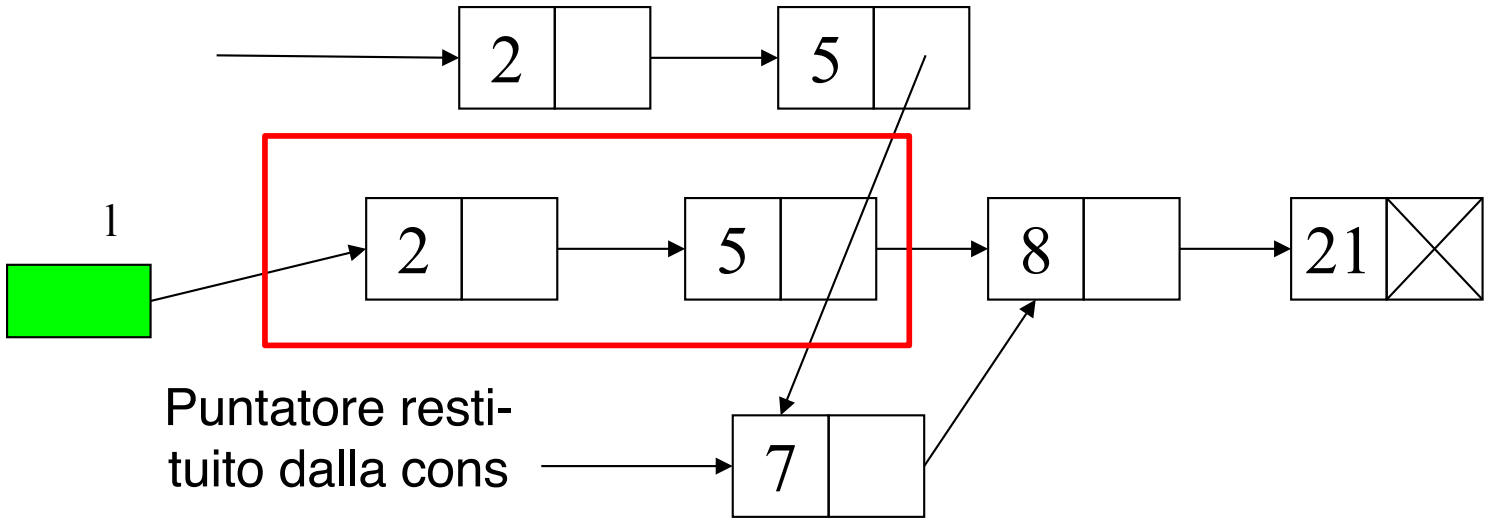
```
showList( );
```

Come si presentano le tre sequenze visualizzate?

La funzione insord ricorsiva (*v*ers. r2)

```
list insord(int el, list l) {
 if (l==NULL) return cons(el, l);
 else if (el<=l->value) return cons(el, l);
  else return cons(l->value,insord(el,l->next));
}
```

Structure copying: parte della struttura dati è replicata (dispendiosa come uso dell'heap!)



Esercizi per Laboratorio

- Scrivere una versione ricorsiva della funzione **`showList`**
- Scrivere una versione ricorsiva della funzione **`member`**

- Scrivere una versione iterativa e una ricorsiva della funzione **`lenght`** che calcoli la lunghezza di una lista
- Scrivere una versione iterativa e una ricorsiva della funzione **`sumlist`** che calcoli la somma degli elementi di una lista di interi
- Definire una funzione **`subList`** che, dato un intero positivo n e una lista l, restituisca una lista che rappresenti *la sotto-lista di quella data a partire dall'elemento n-esimo*

ESEMPIO: l = [1, 13, 7, 9, 10, 1]
subList(2, l) = [7, 9, 10, 1]