Liste concatenate

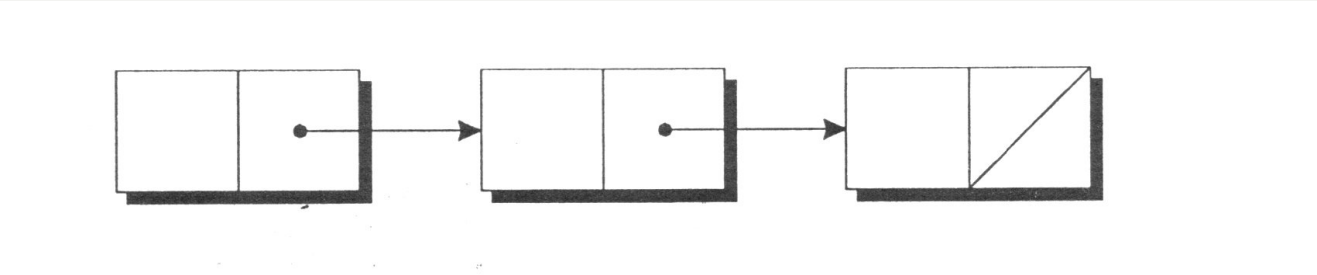
Violetta Lonati

Università degli studi di Milano
Dipartimento di Informatica

Laboratorio di algoritmi e strutture dati
Corso di laurea in Informatica

Liste concatenate

Una lista concatenata consiste di una catena di strutture chiamate nodi. Ogni nodo contiene un puntatore al prossimo nodo della catena. L'ultimo nodo contiene il puntatore nullo (rappresentato da una diagonale).



Struttura nodo

```
struct node {
  int value;              /* dato memorizzato nel nodo */
  struct node *next;      /* puntatore al prossimo nodo */
};
```

NB: non è possibile usare `typedef`, ma è necessario usare un tag, poichè la struttura contiene un puntatore ad una struttura dello stesso tipo.

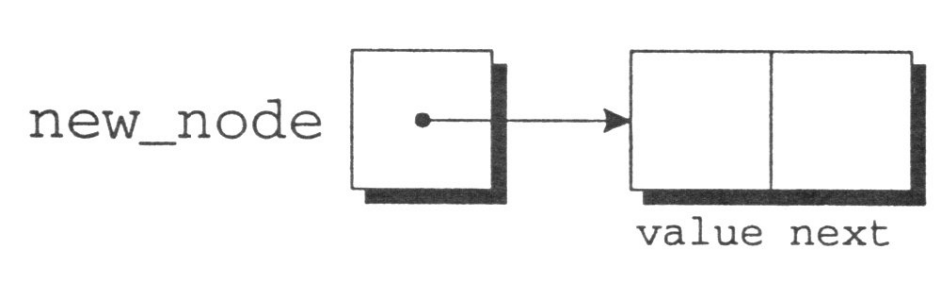
Bisogna tenere traccia di dove la lista comincia. Se la lista è ancora vuota, dichiariamo un puntatore a `struct node` e lo inizializziamo a `NULL`.

```
struct node *first = NULL;
```

Creazione di un nuovo nodo

1. allocare memoria per il nuovo nodo;
2. memorizzare il dato nel nuovo nodo;
3. inserire il nodo nella lista.

```
struct node *new_node;
new_node = malloc( sizeof( struct node ) );
```



Per memorizzare il dato nel nuovo nodo:

```
new_node -> value = 10;
```

oppure

```
scanf( "%d", &new_node -> value );
```

Inserimento in testa

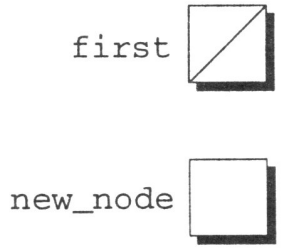
Se `first` punta all'inizio della lista e `new_node` è un nodo (già allocato) da inserire in testa alla lista, basta:

1. fare in modo che il successore di `new_node` sia proprio `first`;
2. impostare `new_node` sia la nuova testa.

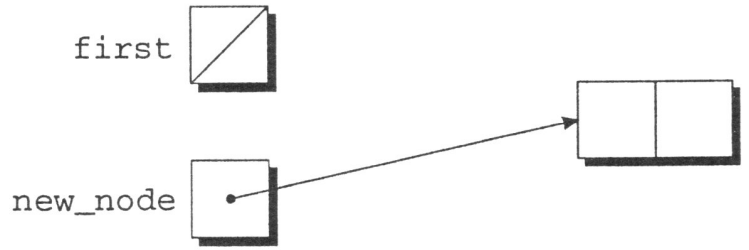
```
new_node -> next = first;
first = new_node;
```

Inserimento in testa - esempio

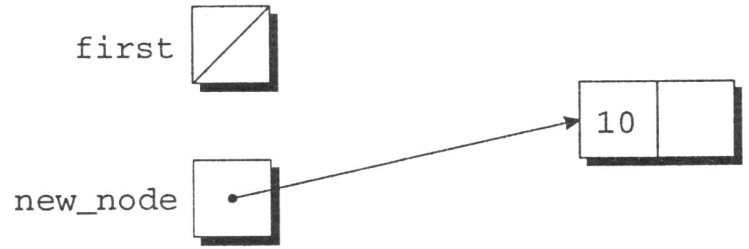
```
first = NULL;
```




```
new_node = malloc(sizeof(struct node));
```




```
new_node->value = 10;
```

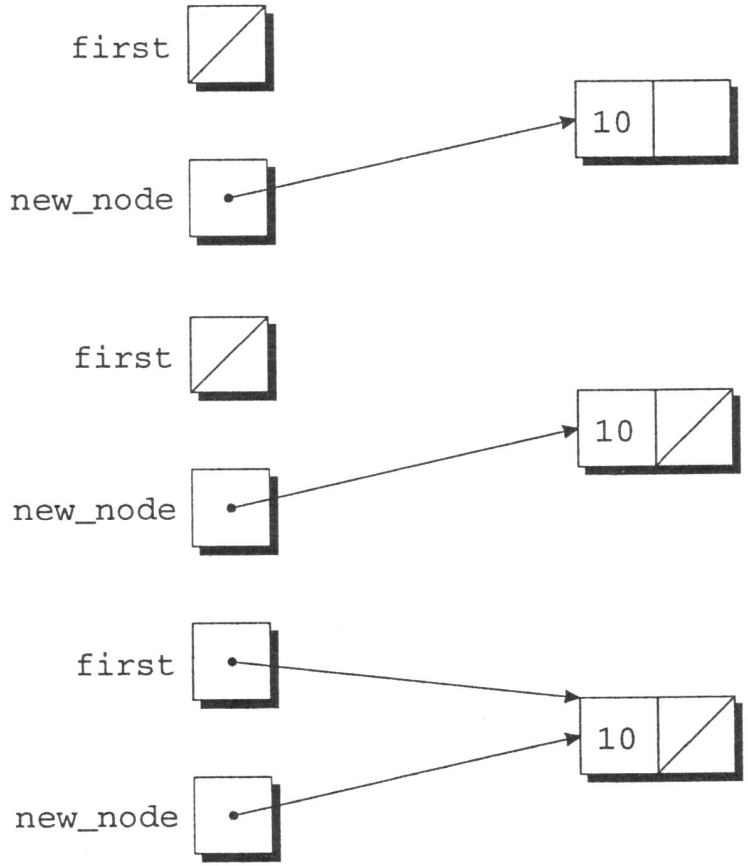


Inserimento in testa - esempio

```
new_node->value = 10;
```

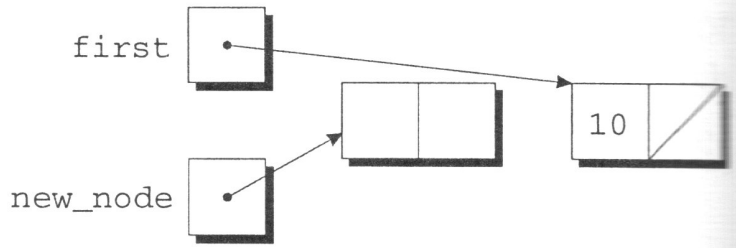
```
new_node->next = first;
```

```
first = new_node;
```

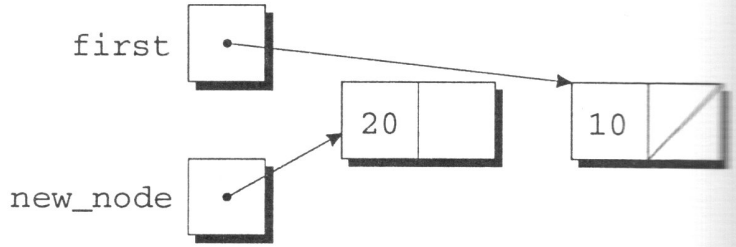


Inserimento in testa - esempio

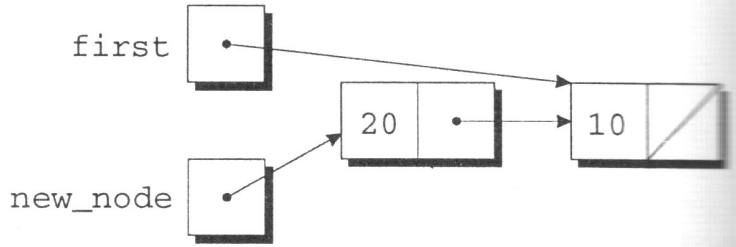
```
new_node = malloc(sizeof(struct node));
```



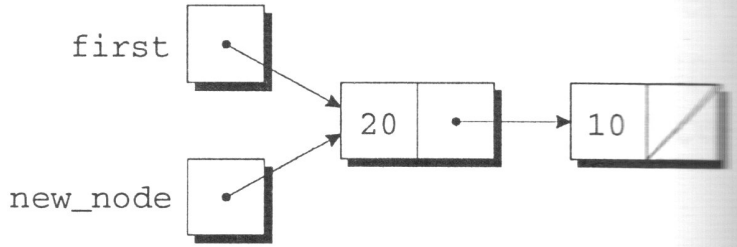

```
new_node->value = 20;
```




```
new_node->next = first;
```




```
first = new_node;
```



Inserimento in testa - continua

Scriviamo una funzione che effettua l'inserimento in testa: `list` sia il puntatore al primo nodo della lista e `n` il nuovo valore da inserire.

```
struct node *add_to_list( struct node *list, int n ){
  struct node *new_node;
  new_node = my_malloc( sizeof( struct node ) );
  new_node -> value = n;
  new_node -> next = list;
  return new_node;
}
```

Ricerca di un nodo

Scorriamo la lista a partire dalla testa cercando il valore desiderato all'interno dei nodi. Usiamo un puntatore che ad ogni passo punta al nodo che stiamo visitando.
Scriviamo una funzione che effettua la ricerca di un elemento in una lista: `list` sia il puntatore al primo nodo della lista e `n` il valore da cercare.

```
struct node *search_list( struct node *list, int n ){
  struct node *p;

  for ( p = list; p!= NULL; p = p -> next )
    if ( p -> value == n )
      return p;
  return NULL;
}
```

Ricerca di un nodo - variante

Elimino puntatore `p` e uso direttamente `list` (tanto è passata per valore!)

```
struct node *search_list( struct node *list, int n ){
  while ( list != NULL && list -> value != n )
    list = list -> next;
  return list;
}
```

Cancellazione di un nodo

1. Trovare il nodo da eliminare;
2. modificare il nodo precedente in modo che punti al nodo successivo a quello da cancellare;
3. liberare con `free` lo spazio occupato dal nodo cancellato.

Per effettuare il punto 1) non mi basta usare un solo puntatore come per la ricerca, perchè una volta trovato, non sappiamo più dov'era il suo predecessore!! Usiamo due puntatori: `cur` punta al nodo corrente; `prev` punta al suo predecessore.

```
for ( cur = list, prev = NULL;
      cur != NULL && cur -> value != n;
      prev = cur, cur = cur -> next )
   ;
```

Cancellazione di un nodo

Per effettuare il punto 2) e il punto 3) bastano le istruzioni:

```
prev -> next = cur -> next;
free( cur );
```

Bisogna però tenere conto dei casi limite, in cui `n` non si trova nella lista, oppure si trova in testa:

```
if ( cur == NULL )
  return list;                          /* n non trovato
if (prev == NULL )
  list = list -> next;                  /* n nel primo n
else
  prev -> next = cur -> next;
free( cur );
return list;
```

Cancellazione di un nodo - esempio

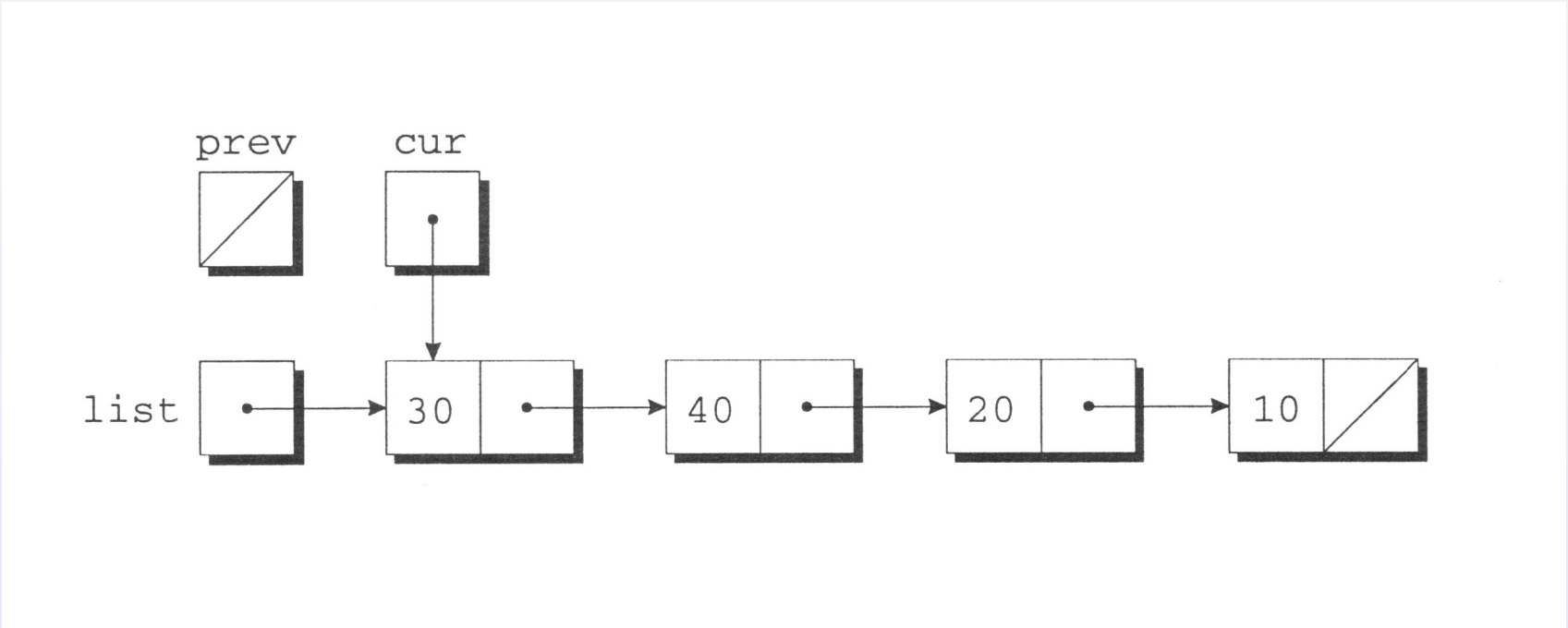
Consideriamo la lista contenente i valori 30, 40, 20 e 10 in quest'ordine e cancelliamo il valore 20.

```
for ( cur = list, prev = NULL;
      cur != NULL && cur -> value != n;
      prev = cur, cur = cur -> next )
   ;
...
prev -> next = cur -> next;
```

Cancellazione di un nodo - esempio

Consideriamo la lista contenente i valori 30, 40, 20 e 10 in quest'ordine e cancelliamo il valore 20.

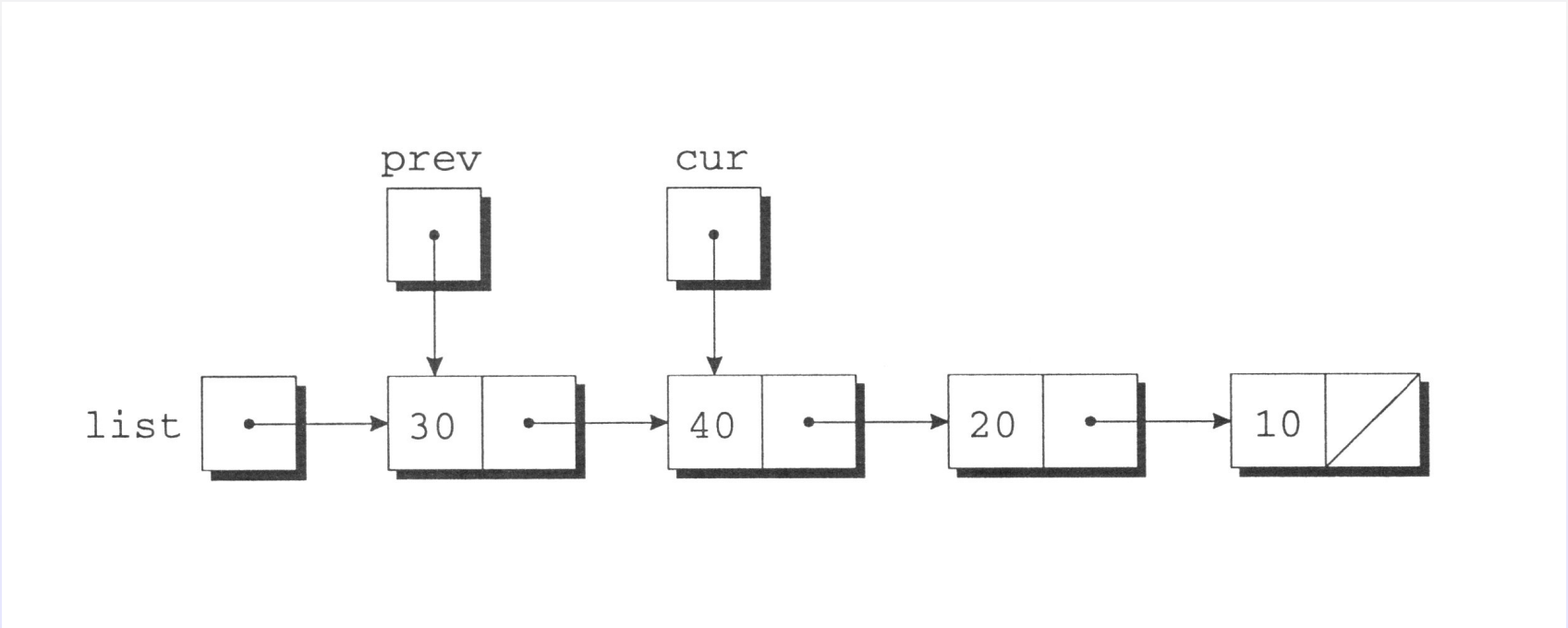
```
for ( cur = list, prev = NULL;
      cur != NULL && cur -> value != n;
      prev = cur, cur = cur -> next )
   ;
...
prev -> next = cur -> next;
```



Cancellazione di un nodo - esempio

Consideriamo la lista contenente i valori 30, 40, 20 e 10 in quest'ordine e cancelliamo il valore 20.

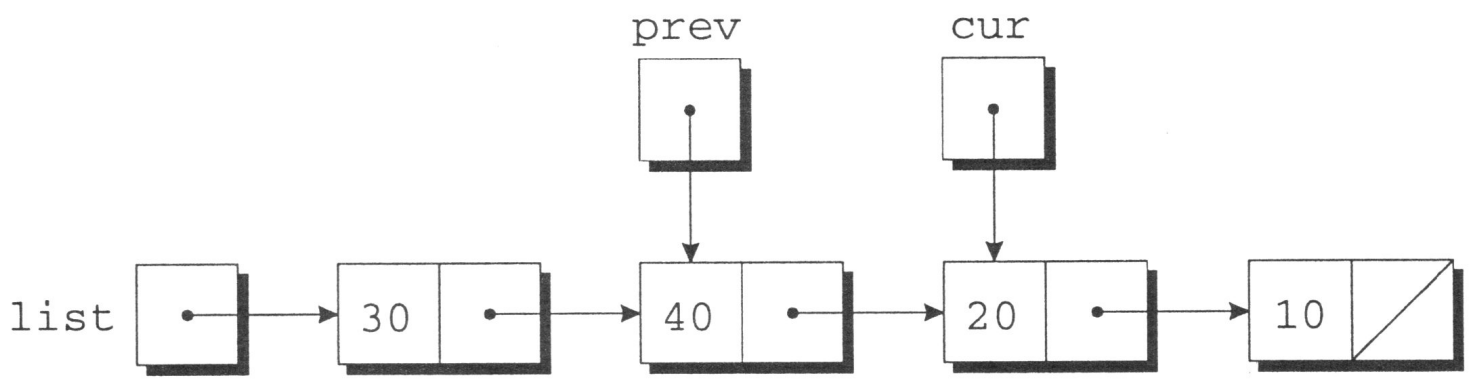
```
for ( cur = list, prev = NULL;
      cur != NULL && cur -> value != n;
      prev = cur, cur = cur -> next )
   ;
...
prev -> next = cur -> next;
```



Cancellazione di un nodo - esempio

Consideriamo la lista contenente i valori 30, 40, 20 e 10 in quest'ordine e cancelliamo il valore 20.

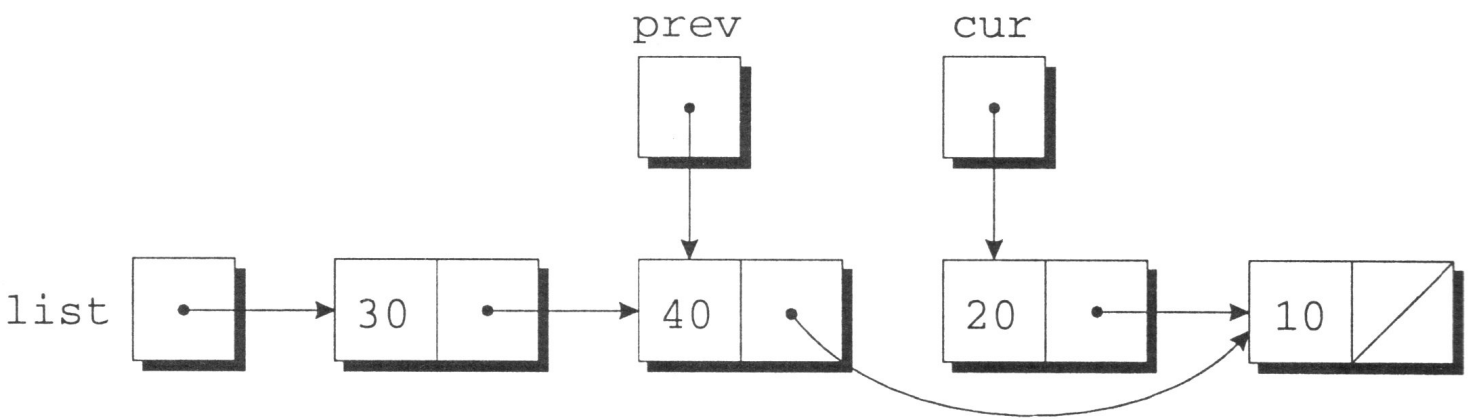
```
for ( cur = list, prev = NULL;
      cur != NULL && cur -> value != n;
      prev = cur, cur = cur -> next )
   ;
...
prev -> next = cur -> next;
```



Cancellazione di un nodo - esempio

Consideriamo la lista contenente i valori 30, 40, 20 e 10 in quest'ordine e cancelliamo il valore 20.

```
for ( cur = list, prev = NULL;
      cur != NULL && cur -> value != n;
      prev = cur, cur = cur -> next )
   ;
...
prev -> next = cur -> next;
```



Liste ordinate

Anzichè effettuare l'inserimento in testa, bisogna prima trovare il punto giusto in cui inserire il nuovo nodo. Scorro la lista usando di nuovo due puntatori `cur` e `prev`, fermandomi quando `cur -> value` diventa maggiore del valore cercato: questo significa che il nodo va inserito fra `prev` e `cur`.