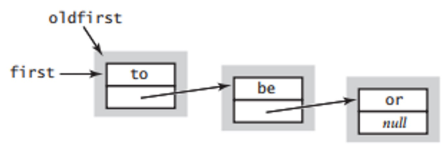
Inserting/removing

Saturday, September 10, 2022 6:15 PM

- Inserting an new node at the beginning of a linked list.

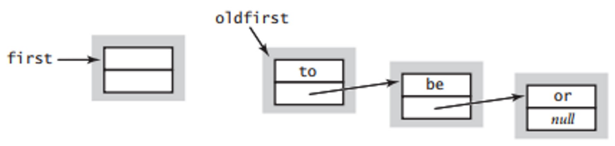
save a link to the list

```
Node oldfirst = first;
```



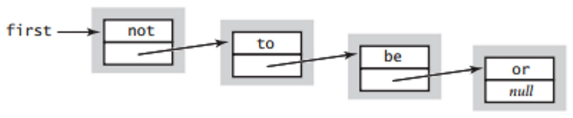
create a new node for the beginning

```
first = new Node();
```



set the instance variables in the new node

```
first.item = "not";
first.next = oldfirst;
```

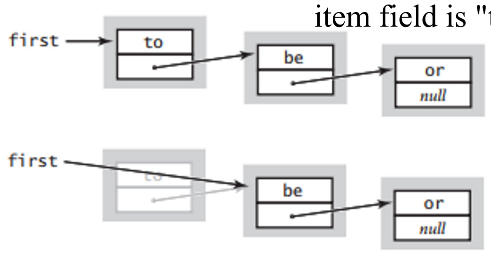


The amount of time to insert a new node at the beginning of a linked list is independent to the length of the list.

- Removing the first node from a list

```
first = first.next;
```

Make the reference of **first** itself to be **the reference of the second node**. As a result, **first** is no longer refers to the first node (whose item field is "to"), instead, **first** is poiniting the second node object.

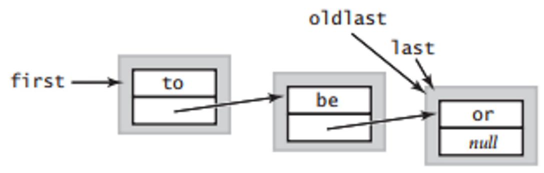


The running time to remove a new node at the beginning of a linked list is independent to the length of the list.

- Inserting at the end of a linked list

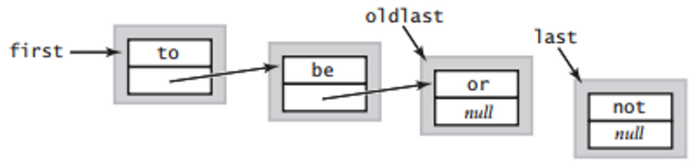
save a link to the last node

```
Node oldlast = last;
```



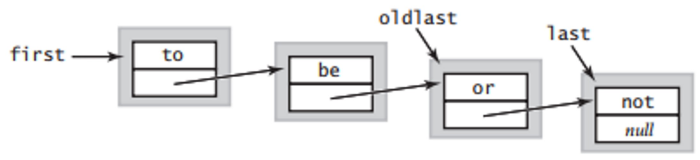
create a new node for the end

```
Node last = new Node();
last.item = "not";
```



link the new node to the end of the list

```
oldlast.next = last;
```



The running time to insert a new node at the end of a linked list is independent to the length of the list.

- Insert/remove at other positions and remove the last mode
    - These actions requires information other than the first and last node.

      For example, to remove the last node, I need to set the next field of the previous node (second to the last node) to null. However, in the absence of information about this node, I can't perform this action.

      In situation like this, one approach is to traverse the entire list looking for the node that links to **last**. Such a solution is undesirable because it takes time proportional to the length of the list.

      The standard solution to enable arbitrary insertions and deletions is to use a double-linked list, where each node has two links, one in each direction.

- Link list loop

```
for (Node x = first; x != null; x = x.next)
{
   // Process x.item.
}
```