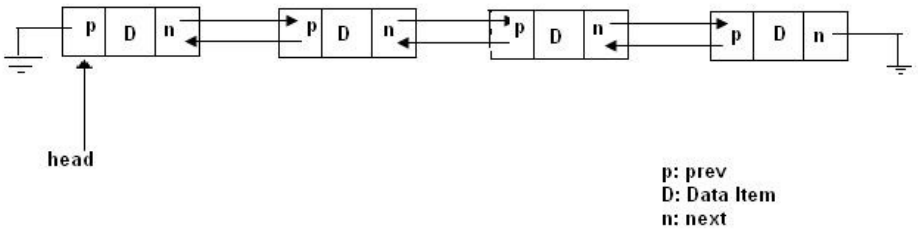
4 April, 2008

Doubly Linked List

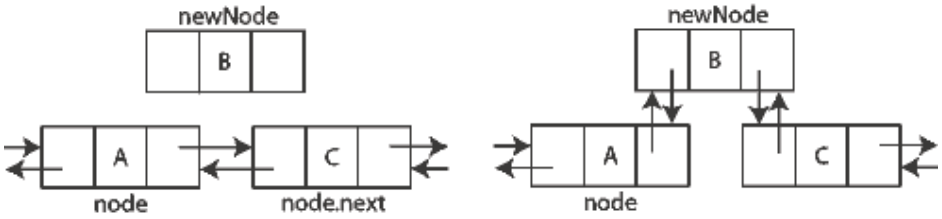


Doubly-linked list(DLL) is a more sophisticated kind of linked list. In DLL, each node has two links: one points to previous node and one points to next node. The previous link of first node in the list points to a Null and the next link of last node points to Null.

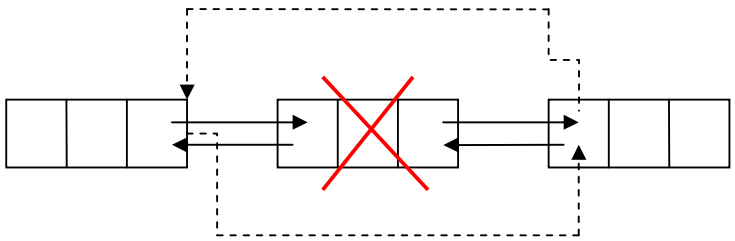
The code of the Double Linked List class is as follows:

```
class Node{
        Item item;
        Node prev, next;

        /* Constructors of Node Class */
        Node(){
                this(null, null, null);
        }

        Node(Item t){
                this(t, null, null);
        }

        Node(Item t, node next, node prev){
                this.item = t;
                this.next = next;
                this.prev = prev;
        }
} // End of class Node


class DLL{
        private Node head = null;

        /* Methods to add an element in the list */
        public void addElementFront(Item t){
                Node n = new Node(t, null, head);
```

```
        if(head!=null)
                head.prev = n;
        head = nl
}
```




```
private void addElementAfter(Node n, Item t){
        Node newN = new Node(t, n, n.next);
        if(n.next != null)
                newN.next.prev = newN;
        n.next = newN;
} // This method is made private to avoid the external access to Node n
public void addElementRear(Item t){
        Node n = head;
        if(n == null){
                addElementFront(t);
                return;
        } //If the list is empty, then we can front and rear element after \
                addition of element will be same

        // Traverse through the list till last element is encountered
        while(n.next != null)
                n = n.next;

        // Now add element after the current last element of list
        addElementAfter(n, t);
}
```



```
/* Method to delete an element from the list */
private void deleteElement(Node n){
        if(n.prev != null){
```

```
                    n.prev.next = n.next;
                if(n.next != null)
                    n.next.prev = n.prev;
        }
} //End of class DLL
```

Compared to Singly Linked List, in Doubly Linked List, one has to update more pointers, but less information is required as one can use previous links to observer preceding elements in the list.

***Circular Linked List*:**

Circular Linked List is a special type of linked list in which all the nodes are linked in continuous circle. Circular list can be Singly or doubly linked list. Note that, there are no Nulls in Circular Linked Lists. In these type of lists, elements can be added to the back of the list and removed from the front in constant time.

Both types of circularly-linked lists benefit from the ability to traverse the full list beginning at any given node. This avoids the necessity of storing first Node and last node, but we need a special representation for the empty list, such as a last node variable which points to some node in the list or is *null* if it's empty. This representation significantly simplifies adding and removing nodes with a non-empty list, but empty lists are then a special case.

Circular linked lists are most useful for describing naturally circular structures, and have the advantage of being able to traverse the list starting at any point. They also allow quick access to the first and last records through a single pointer (the address of the last element).

Typecasting refers to changing an entity of one data type into another.

e.g.
```
byte b;
(int) b;
```

In object-oriented programming, I**nheritance** is a way to form new classes using classes that have already been defined. The new classes, known as *derived classes*, take over (or *inherit*) attributes and behavior of the pre-existing classes, which are referred to as *base classes*. It is intended to help reuse existing code with little or no modification.

In the next lecture, we will understand the Object Oriented concepts like Class Inheritance and will look at some examples of Type Casting.