BENNETT UNIVERSITY
TIMES OF INDIA GROUP

ECSE207L
DATA STRUCTURES

Dr. Tapas Badal
Dept. of CSE
Bennett University

Recursive Definition of List Node

```
public class Node {
        public int data;
        public Node link;

         Node(int n, Node p) {
                 data = n;
                 link = p;
         }
 };
```

Extended Definition

```
public class Node {
        public int data;
        public Node link;

        Node() {                    // a simple node
                item = 0;
                link = null;
        }

         Node(int n) {      // a node with a given value
                data = n;
                link = null;
        }

         Node(int n, Node p) {// a node with given value and reference
                data = n;
                link = p;
         }
 };
```

getLink, setLink, getData

```
•public Node getLink()
  {
        return link;
  }

• public void setLink(Node n)
  {
        link = n;
  }

•public int getData()
  {
        return data;
  }
```

Inserting a node in a Linked List

Inserting a node at front of the list

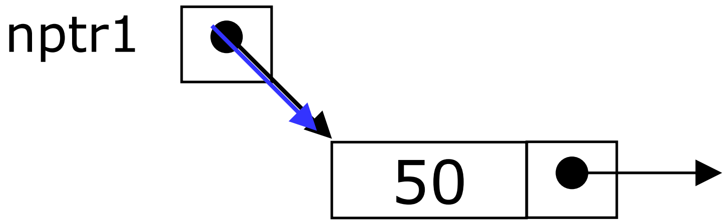
- Assume we have a list pointed by Front.
- Create a new node nptr. Set its data with given value.
- If the Linked list is initially empty, the new node becomes the Front node.
- Otherwise set the link of nptr to Front.
- declare Front to be the new pointer to the list.

```
Node nptr = new Node ( 50, null);
nptr.setLink(Front); //attach old Front as link to new node nptr
 Front = nptr;              // declare new node as Front node
```

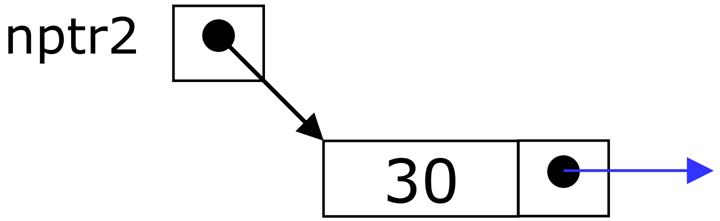
Creating linked list with two nodes

- Let us create a node with the statement:

```
Node nptr1 = new Node(50, null);
```



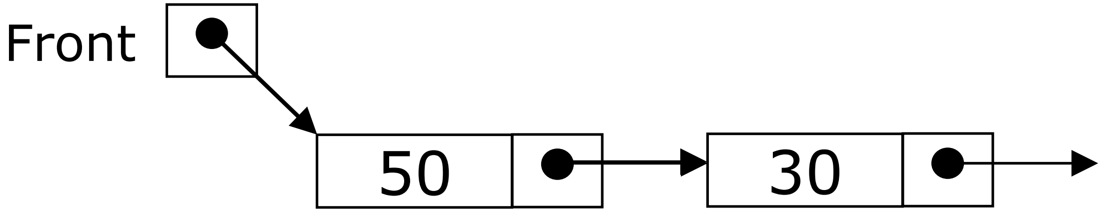
- Similarly create another node `Node nptr1 = new Node(30, null);`



- Let us now form a linked list with first node being the Front node

Linked list with first node followed by second node

- Front = nptr1
- Front.setLink (nptr2)
- This statement will set the link value of first pointer with nptr2
- Since nptr2 points to second node, they get linked

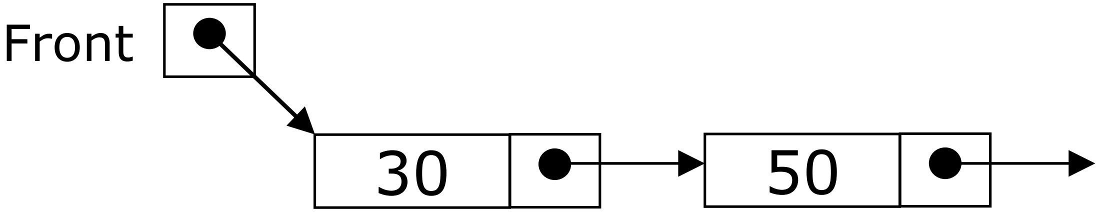


Linked list with second node followed by first node

- Front = nptr2

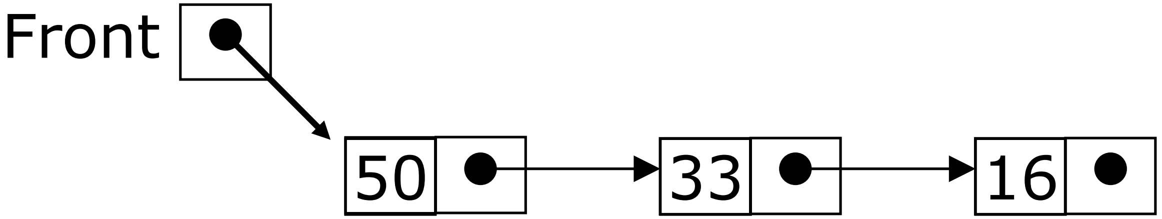
  Front.setLink (nptr1)

- This statement will set the link value of second node to nptr1
- Since nptr1 points to second node, they get linked as shown



Inserting the new node at front of List

- Suppose we wanted to insert a node containing value 82 at front of the linked list.



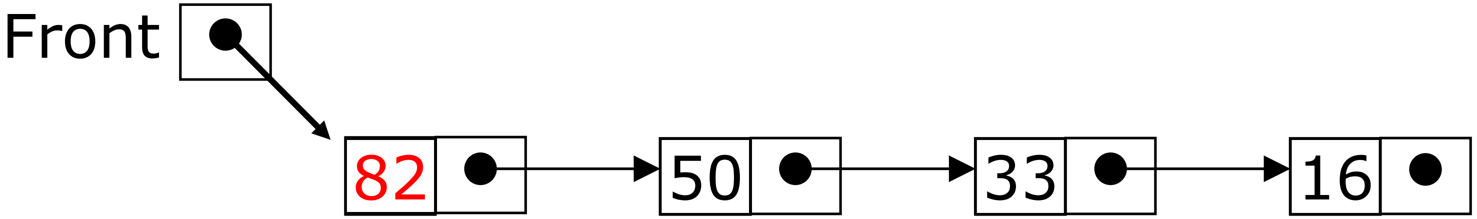
- First form a new node with value 82 (and pointing to null) Next we link node 82 to Front (node containing 50)

- Finally, we declare node 82 to be the new Front node

```
Node nptr = new Node ( 82, null);
if (Front == null)
    Front = nptr;
else
   {
     nptr.setLink(Front);
          Front = nptr;
    }
```

Modified linked list

- The linked list now takes the form:



Traversing a Linked list with a pointer

- Set a running pointer ptr to Front.
- Use method getData to read the value in that node
- Set pointer to next node by using method getLink.
- Keep on traversing till link is null (list is over).

```
Node ptr = Front;
while (ptr != null)
        {
            System.out.print(ptr.getData()+ “  ”);
             ptr = ptr.getLink();
        }
```

Count number of nodes

- Set a running pointer ptr to Front. Start a counter. Traverse through the list, and increment the counter until link reaches null value. Then print value of the counter. Add code for handling an empty list.

```
Node ptr = Front;
int count = 0;
while (ptr != null)
      {
                count++;
                 ptr = ptr.getLink();
      }
      System.out.println("nodes ="+ count);
```

Recursive methods in linked lists

- Now we study recursive methods which operate on Linked Lists.
- This is natural as linked lists themselves are recursive data structures.
- Recursive methods operating on linked lists are often simpler to write and easier to understand than their iterative counterparts.
- Later when we study Tree data structure, we shall see that some methods can only be written using recursion.

Recursively print contents of nodes

- Set a running pointer ptr to Front. Print the data for this node and then recursively call the function with link to current ptr.

```
Public  static void  printList ( Node ptr )
{
        if (ptr != null)
        {
                    System.out.print(ptr.getData()+ “  ”);
                     printList( ptr.getLink() );                              //recursive call
          }
      }
```

Recursively count number of nodes

- If pointer is not null, add 1 to nodes in the remaining list starting with the next node. When the list reaches the end, ptr reaches null, which adds zero to the count.

```
Public  static int countList ( Node ptr )
{
       if (ptr == null)                              // base case
           return 0;
       else
            return 1 + countList( ptr.getLink() );   //continue case
}
```