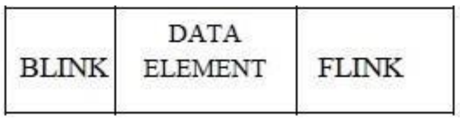
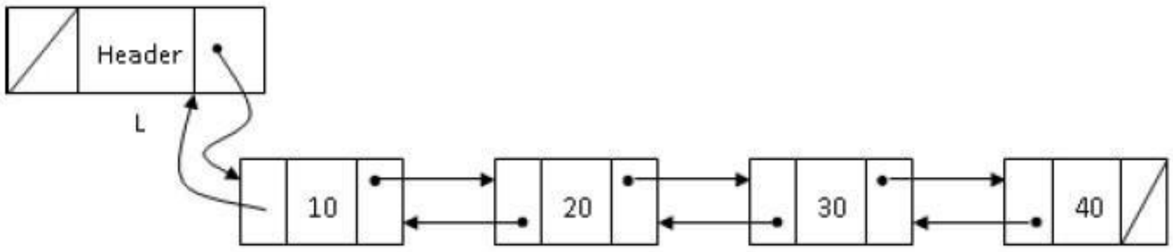
2.7 DOUBLY LINKED LIST

- A Doubly linked list is a linked list in which each node has three fields namely data field, forward link (FLINK) and Backward Link (BLINK).
- FLINK points to the successor and BLINK points to the predecessor.





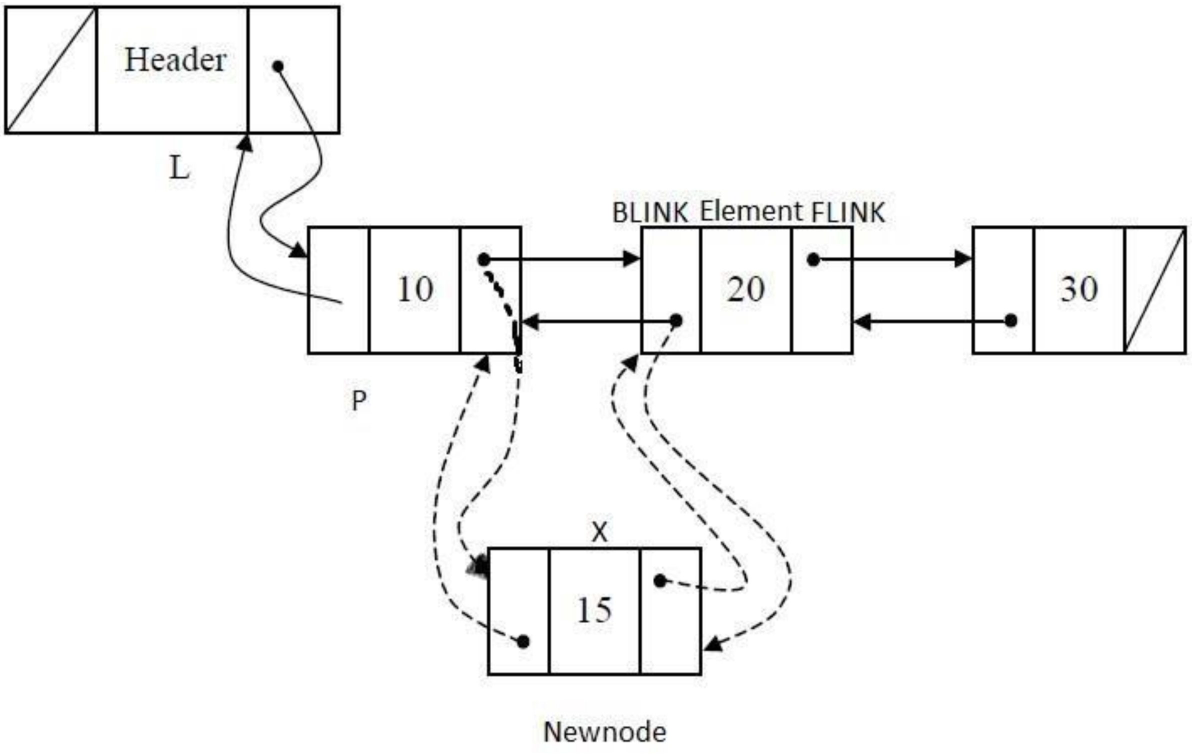
Node declaration

class Node**:**

```
class Node

{

public:

    int data;

    Node *prev;

    Node *next;

};

class DoublyList

{

    Node *head;

public:
    DoublyList()
    {
        head = NULL;
    }
};
```

INSERTION:

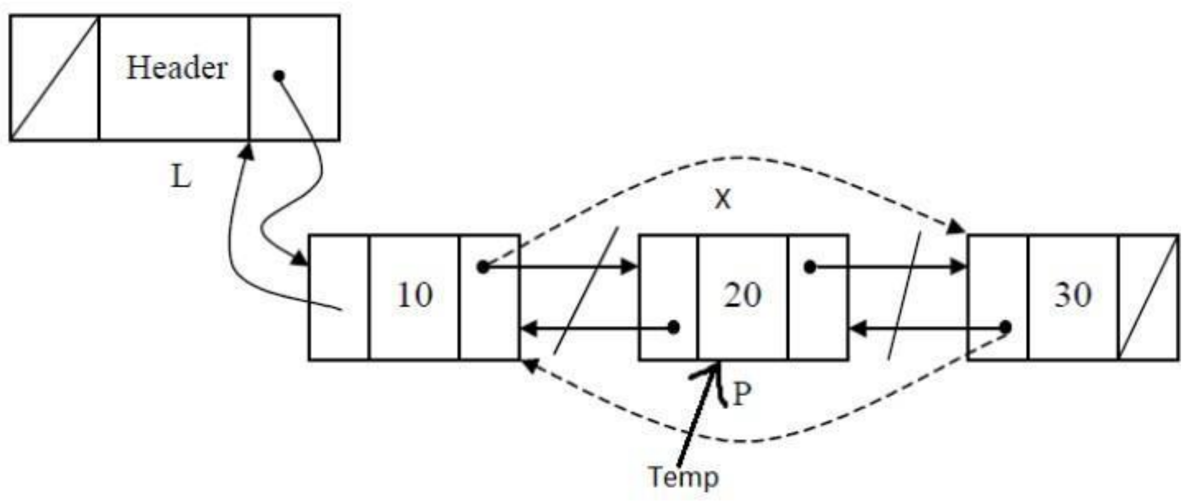
1. Obtain space for new node.
2. Assign data to the data field of the new node.
3. Assume current node P as the node after which we wish to insert.
4. Set the FLINK field of the new node to the successor node
5. Set the FLINK of current node and BLINK of successor node to newnode.
6. Set the BLINK of newnode to current node



```
void insert(int item)
  {
    Node *newNode = new Node;
    newNode->data = item;
    newNode->prev = NULL;
    newNode->next = NULL;
    if (head == NULL)
    {
      head = newNode;
    }
    else
    {
      Node *temp = head;
      while (temp->next != NULL)
        temp = temp->next;

      temp->next = newNode;
      newNode->prev = temp;
    }
  }
```

DELETION:

1. Find the node to be deleted and assume it as current node P
2. Set the FLINK of predecessor node as the FLINK of current node
3. Set the BLINK of successor node as the BLINK of the current node
4. Free the memory used by current node



```
void del(int item)
{
    Node *temp = head;

    while (temp != NULL && temp->data != item)
        temp = temp->next;

    if (temp == NULL)
    {
        cout << "\nElement not found";
        return;
    }
    if (temp->prev != NULL)
        temp->prev->next = temp->next;
    else
        head = temp->next;
    if (temp->next != NULL)
        temp->next->prev = temp->prev;

    delete temp;
    cout << "\nElement deleted";
}
```

SEARCHING:

- Start from the first node and compare the data part with element to be searched
- If it is present, then the corresponding position is returned.

```
void search(int item)
{
    Node *temp = head;
    int pos = 1;
    while (temp != NULL)
    {
        if (temp->data == item)
        {
            cout << "\nElement found at position: " << pos;
            return;
        }
        temp = temp->next;
        pos++;
    }
    cout << "\nElement not found";
}
```

Advantages

- Deletion operation is easier.
- Finding the predecessor & successor of a node is easier.

Disadvantages

More memory space is required than singly linked list, since it has two pointers.