**DOUBLY LINEAR LINKED LIST**

Doubly linked list is a data structure that consists of a set of sequentially linked records called nodes. Each node contains two fields that are reference to previous and next node in sequence of nodes. In singly linear linked list only forward traversing is possible as it contains only next field but in doubly linear linked list traversing is bidirectional, can traverse in both forward and backward direction.

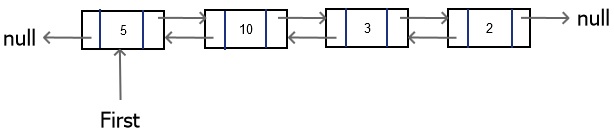


Fig: Doubly Linear Linked List

**Algorithms for operations in Doubly Linear Linked List**

* **Insertion at the beginning**

Insert\_front(head,item)

1. Start
2. Allocate memory for a node, n=new Node and set n->data=item
3. If head==NULL set head=n
4. Else create new pointer as temp=head and set n->next=temp, temp->prev=n and head=n
5. Exit

* **Insertion at the end**

Insert\_back(head,item)

1. Start
2. Allocate memory for a node, n=new Node and set n->data=item
3. If head==NULL set head=n
4. Else set temp=head and traverse temp until temp->next!=NULL then set temp->next=n and set n->prev=temp
5. Exit

* **Insertion after a specific node**

Insert\_after(head,data,after)

1. Start
2. If node n with n->data=after doesn’t exist then display node not found and exit
3. Else goto step 4
4. Set temp=head and traverse the temp until temp->data!=after
5. Set n->next=temp->next, n->prev=temp,temp->next->prev=n andtemp->next=n
6. End if
7. Exit

* **Insertion before a specific node**

Insert\_before(head,data,before)

1. Start
2. If node n with n->data=before doesn’t exist then display node not found and exit
3. Else goto step 4
4. Set temp=head and traverse the temp until temp->data!=before
5. Set n->next=temp,n->prev=temp->prev,temp->prev->next=n and temp->prev=n
6. End if
7. Exit

* **Deletion at the beginning**

Remove\_front(head)

1. Start
2. Set ptr=head
3. If ptr==NULL then display empty list and exit else goto step 4
4. If ptr->next==NULL then set head=NULL else goto step 5
5. Set temp=head and temp->next->prev=NULL and head=temp->next
6. Delete ptr
7. Exit

* **Deletion at the end**

Remove\_back(head)

1. Start
2. If head==NULL display empty list and exit else goto step 3
3. Set ptr=head and traverse the ptr until ptr->next!=NULL
4. Set temp=ptr->prev, temp->next=NULL and ptr->prev=NULL
5. Delete ptr
6. Exit

* **Deletion after a specific node**

Remove\_after(head,after)

1. Start
2. If node with n->data==after doesn’t exist then display no node and exit
3. Else set ptr=head and traverse the ptr until ptr->data!=after
4. If ptr->next==NULL then display No item to delete and exit
5. Else if ptr->next->next==NULL call remove\_back(head)
6. Else set temp=ptr->next, ptr->next=temp->next and temp->next->prev=ptr and delete temp
7. End inner if
8. End outer if
9. Exit

* **Deletion before a specific node**

Remove\_after(head,before)

1. Start
2. If node with n->data==before doesn’t exist then display no node and exit
3. Else set ptr=head and traverse the ptr until ptr->data!=before
4. If ptr->prev=NULL then display no item to delete and exit
5. Else if ptr->prev->prev==NULL then call remove\_front(head)
6. Else set temp=ptr->prev, ptr->prev=temp->prev, temp->prev->next=ptr and delete temp
7. End inner if
8. End outer if
9. Exit