Doubly Linked Lists (DLL)

A doubly linked list is also called as **two-way linked list.** It consists of data as well as links to the next item and also previous item shown in below figure (1).

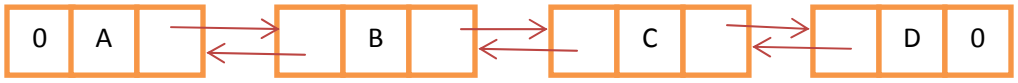


Fig: Doubly Linked List

As shown in the above figure, each item has two links or pointers. One link (forward) points to the next item in the list, and the second link (backward) points to previous item in the list. Whereas first item in the list has points back to null, because this is the very first item in the list and the last (tail node) item points next to null (dummy value of '0'), as all other pointers in the list points to the next and previous items.

Type declaration of doubly linked list as shown below:

```
struct DLL{

int item;

struct DLL *next;

struct DLL *previous;

}
```

Advantages

It has two advantages over singly linked list:

1. We can traverse (or navigate) list in both directions.
   i) We can start with the first item in the list and read up to the last item in the list in forward direction. For example given above list read elements in the order A-B-C_D.
   ii) We can start with the last item in the list and read up to first item in the list in backward direction. For example, the given above list reads elements in the order D-C-B_A.
2. If any failure causes with one of the links (either forward or backward) to become invalid, this can be reconstructed with the help of other link. It is not possible with singly linked list because it has only one link.

Disadvantages

1. Each node requires extra pointer, it takes more space.

2. The insertion or deletion of the node takes a bit longer because of more pointer operations.

Doubly Linked List Operations

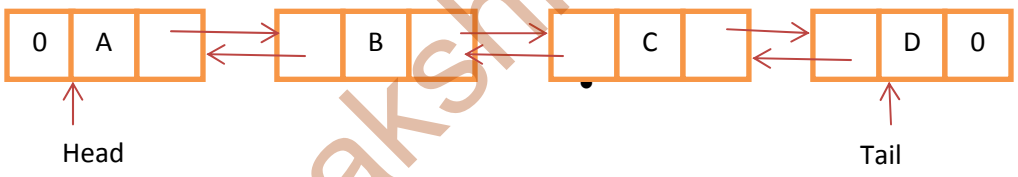
Creating linked list is similar to singly linked list but here two links maintained instead of just one link. Following operations consider as DLL operations.

1. Inserting an item.
2. Deleting an item.
3. Traversing all items in the list.

Traverse the list

Let us assume that head and tail points to the first and last items in the list. We can traverse the list in two ways.

- Start from head follow the pointers and display or count the all items (in forward direction) in the list till reach the tail.
- Start from tail follow the pointers and display or count the all items (in backward direction) in the list till reach the head.



DLL Insertion

Inserting of an item into DLL has three cases same as singly linked list.

- Inserting new node before the head.
- Inserting new node after the tail
- Inserting new node at the middle of the list.

Inserting new node before head in DLL:

The new node next pointer points to the current head node and the current head node previous pointer points to the new node. To make the new node previous pointer to NULL, update the new node as head node.

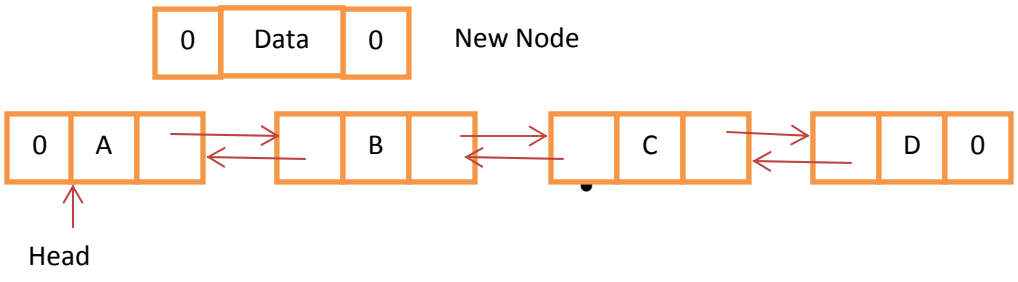


Fig: Before Insertion

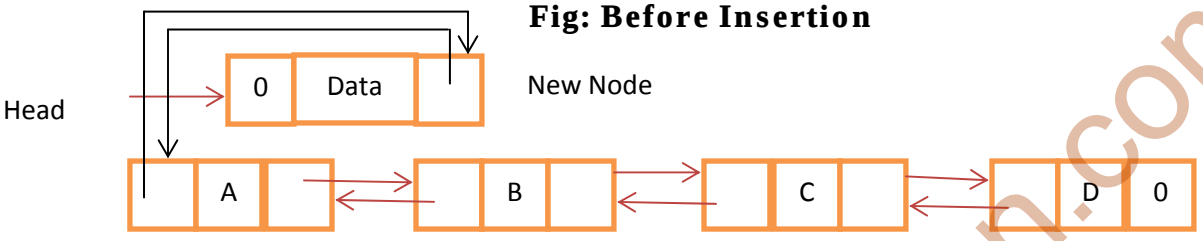


Fig: After Insertion

Inserting new node after tail in DLL

The new node previous pointer points to the current tail node and the current tail node of the next pointer points to the new node. To make the new node next pointer to NULL, update the new node as tail node.

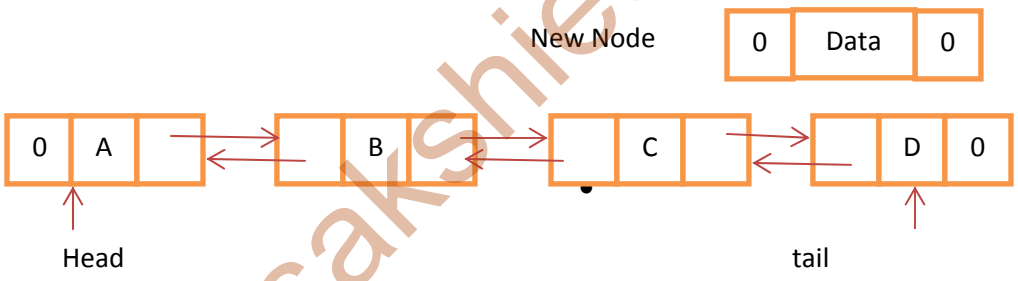


Fig: Before Insertion

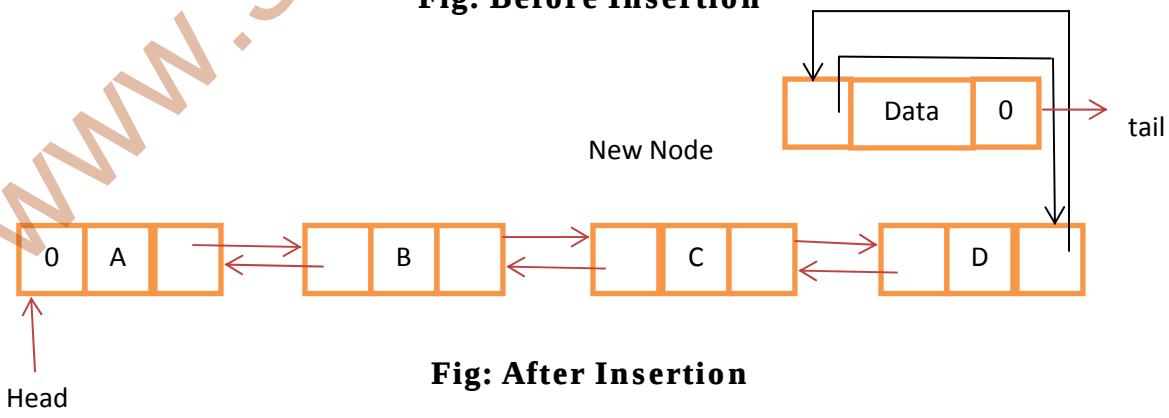


Fig: After Insertion

Inserting the new node in middle of list:

- Traverse the list until you reach the position node and then insert the new node.
- The new node next pointer points to the next pointer of the position node and the right pointer points to the position node.
- Position node next pointer points to the new node and the next node of position node previous pointer points to the new node.

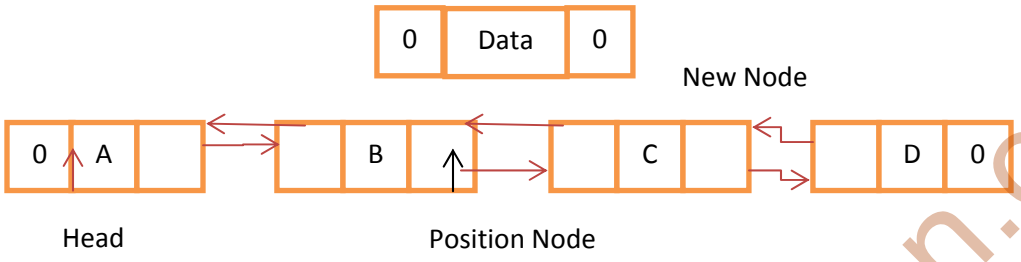


Fig: Before Insertion

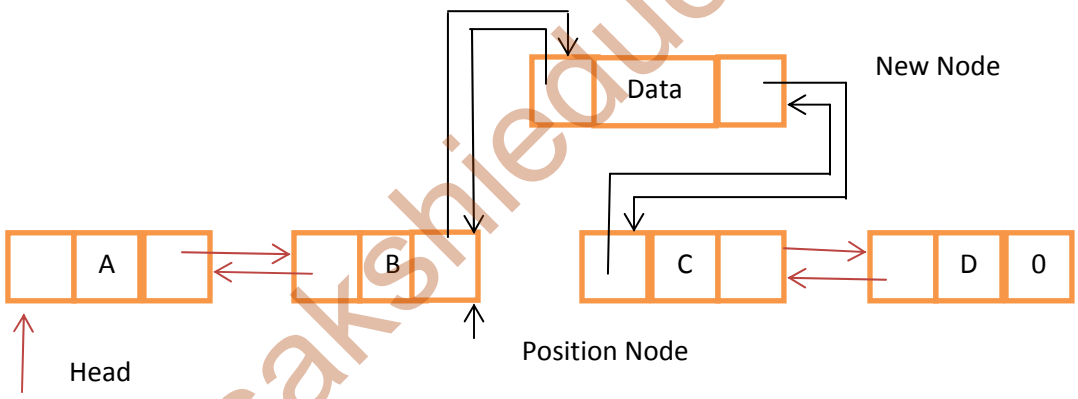


Fig: After Insertion

DLL Deletion

Deleting an item into DLL has three cases same as singly linked list.

- Deleting node at beginning of the list.
- Deleting node at the end of the list
- Deleting node at the middle of the list.

Deleting node at beginning of the list in DLL

Create a temporary node which will points to same head node. Now move the head node's pointer to the next and change the head's previous pointer to NULL. Now delete the temporary node and free the memory.

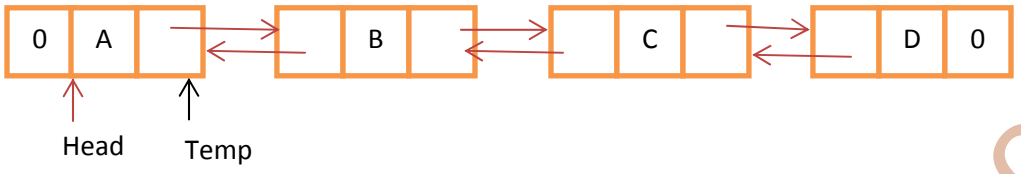


Fig: Before Deletion

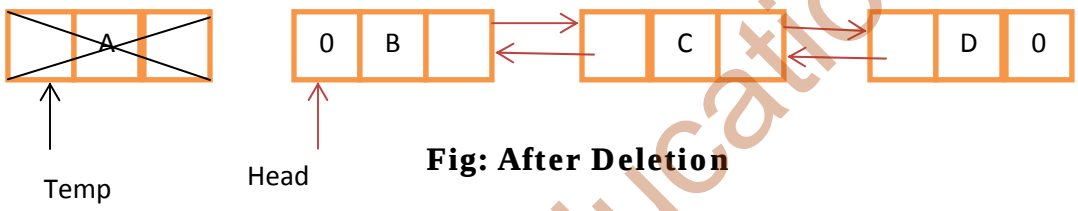


Fig: After Deletion

Deleting node at end of the list in DLL

First traverse the list to find the tail node. While traversing the list we should have to maintain previous node details also. Update the tail previous node next pointer to NULL and dispose the tail node.

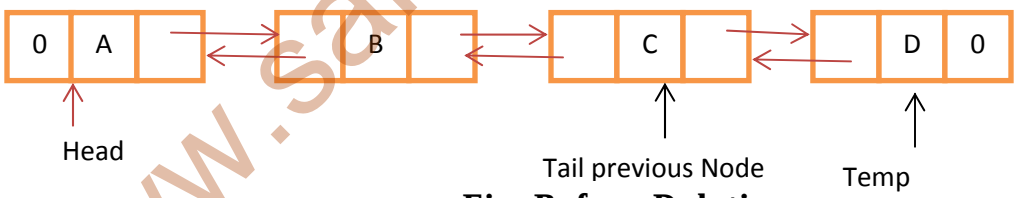


Fig: Before Deletion

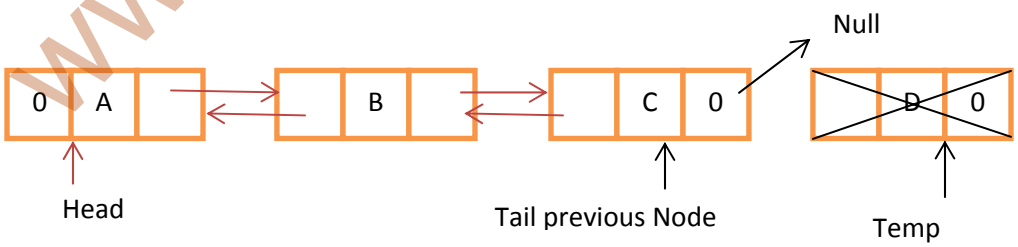


Fig: After Deletion

Deleting the node in middle of list

Traverse the list first to reach the position node at which node to be deleted. Update this position node next pointer with next pointer of node to be deleted and the next node of the previous pointer with previous pointer of node to be deleted. Dispose the memory of node to be deleted.

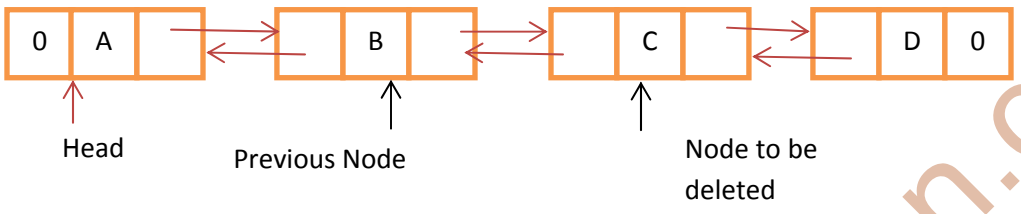


Fig: Before Deletion

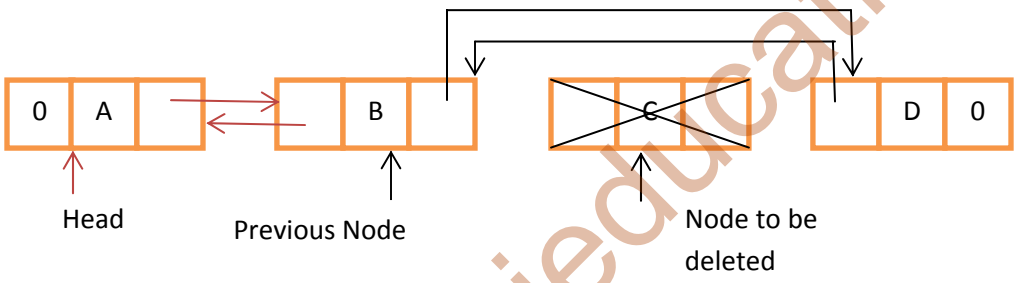


Fig: After Deletion

C Program:

```
//header files
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
//global linked list variables
struct node
{
int data;
struct node *next; //pointer poins to next item
    struct node *previous;//pointer points to previous item

}*head, *tail; //pointer to the first and last entry in the list

/*Function declarations and definitions*/

//Accept and store an item at beginning of the list
voidinsertAtBegining(int value)
{
struct node *newnode;
newnode=(struct node *)malloc(sizeof(struct node));//create and allocate memory to new node
```

```
newnode->data=value;
        //if this is very first item inserted into the list
if(tail==NULL)
  {
newnode->previous=NULL;
newnode->next=NULL;
head=newnode;
                tail=newnode;
  }
else
  {
                newnode->previous=NULL;
newnode->next=head;
                head->previous=newnode;
                head=newnode;
  }
}
//Accept and store an item at end of list.
voidinsertAtEnd(int value)
{
struct node *p,*newnode;
newnode=(struct node *)malloc(sizeof(struct node));//create and allocate memory to temp node
newnode->data=value;
        //if this is very first item inserted into the list
if(tail==NULL)
  {
newnode->previous=NULL;
newnode->next=NULL;
head=newnode;
                tail=newnode;
  }
else
  {
newnode->next=NULL;
                tail->next=newnode;
                newnode->previous=tail;
                tail=newnode;
  }
}
//accept and store an item in middle of list at user selected position
voidinsertAtMiddle(int value, intloc)
{
struct node *newnode,*temp;
        int k=1;
        int n=count();
newnode=(struct node *)malloc(sizeof(struct node));
newnode->data=value;
        //if this is very first item inserted into the list
if(tail==NULL)
  {
newnode->previous=NULL;
newnode->next=NULL;
head=newnode;
                tail=newnode;
                return;
  }
        //check the position exist or not
        if(loc>n){
                printf("length of list is small\n");
                return;
```

```
        }
else
   {
temp=head;
while(temp->next!=NULL && k<loc-1)
        {
temp=temp->next;
                        k++;
        }
if(temp==NULL)
        {
printf("\n%d is not present in list ",loc);
        }
else
        {
                                newnode->next=temp->next;
                                newnode->previous=temp;
                                temp->next->previous=newnode;
                                temp->next=newnode;

        }
   }

}
//Delete an item at beginning of the list
voiddeleteAtBegining(){
        struct node *temp;
temp=head;
        if(tail==NULL){
                //printf("list is empty\n");
                return;
         }
elseif(temp->next==NULL)
   {
                printf("\nData deleted from list is %d \n",temp->data);
free(temp);
head=NULL;
tail=NULL;

   }
        else {

                head=temp->next;
                head->previous=NULL;
                printf("Data deleted from list is %d \n",temp->data);
                free(temp);
         }

}
//Delete an item at end of the list
voiddeleteAtEnd()
{
struct node *t;
   t=tail;
        if(tail==NULL){
                //printf("list is empty\n");
                return;
         }
elseif(t->previous==NULL)
   {
```

```
                printf("\nData deleted from list is %d \n",t->data);
                free(t);
head=NULL;
tail=NULL;

}else{
                tail=t->previous;
                tail->next=NULL;
                printf("\nData deleted from list is %d \n",t->data);
                free(t);
                t=NULL;
        }
}
//Delete an item at middle of the list at user selected position
voiddeleteAtMiddle(intpos)
{
struct node *te,*p;
        int k=1;
int n=count();
        if(tail==NULL){
                //printf("list is empty\n");
                return;
        }
if(head->next==NULL){
                printf("\nData deleted from list is %d \n",te->data);
                free(te);
head=NULL;
tail=NULL;
return;
  }
        //check whether list is small or not
        if(pos>n){
                printf("length of list is small\n");
                return;
        }
        //if position is 1 insert at beginning of list
        if(pos==1){
                deleteAtBegining();
                return;
        }
        else{
                te=head;
                while((te->next!=NULL) && (k<pos-1))
                {
                        te=te->next;
                        k++;
                }
                p=te->next;
                //if position is last delete at tail
                if(p->next==NULL){
                        te->next=NULL;
                        tail=te;
                        printf("data deleted from list is %d\n",p->data);
                        free(p);

                }else{
                        te->next=p->next;//delete at middle position
                        p->next->previous=te;
                        printf("data deleted from list is %d\n",p->data);
                        free(p);
```

```
                        p=NULL;
                    }
            }
            return;
}
//display the items in the list
void display()
{
struct node *temp;
temp=head;
if(temp==NULL)
   {
printf("List is Empty");
   }
while(temp!=NULL)
   {
printf("-> %d ",temp->data);
temp=temp->next;
   }
}
//Return the number of elements in the list
int count()
{
struct node *temp;
        int p=0;
temp=head;
if(temp==NULL)
   {
printf("List is Empty");
                return 0;
   }
while(temp!=NULL)
   {
p++;
temp=temp->next;

   }
        return p;
}
//Main starts here
int main()
{
int value, i, loc;
head=tail=NULL;//initialize head and tail pointers here
printf("Select the choice of operation on link list");
printf("\n1.) Insert at begning\n2.) Insert at end\n3.) Insert at middle");
printf("\n4.) Delete from begining\n5.) Delete from end\n6.) Delete from middle\n7.) Display list\n8.) exit");
while(1){
printf("\nEnter the choice of operation you want to do ");
scanf("%d",&i);
switch(i){
case 1:
      {
                        printf("Enter the value you want to insert in node ");
                        scanf("%d",&value);
                        insertAtBegining(value);
                        display();
                        break;
      }
case 2:
```

```
        {
                        printf("Enter the value you want to insert in node at tail ");
                        scanf("%d",&value);
                        insertAtEnd(value);
                        display();
                        break;
        }
case 3:
        {
                        printf("Enter position at which you want to insert data ");
                        scanf("%d",&loc);
                                printf("enter the data you want to insert in list ");
                                scanf("%d",&value);
                                insertAtMiddle(value,loc);
                                display();
                                break;
        }
                case 4:
        {
                                deleteAtBegining();
                                display();
                                break;
        }
case 5:
        {
                                deleteAtEnd();
                                display();
                                break;
                 }
case 6:
        {
                                printf("Enter the position you want to delete");
                                scanf("%d",&value);
                                deleteAtMiddle(value);
                                display();
                                break;
        }
case 7:
        {
                                display();
                                break;
        }
case 8:
        {
exit(0);
break;
        }
    }
  }

getch();
}
```

Output:

```
1.) Insert at begning
2.) Insert at end
3.) Insert at middle
4.) Delete from begining
5.) Delete from end
6.) Delete from middle
7.) Display list
8.) exit
Enter the choice of operation you want to do 1
Enter the value you want to insert in node 10
-> 10
Enter the choice of operation you want to do 2
Enter the value you want to insert in node at tail 20
-> 10 -> 20
Enter the choice of operation you want to do 1
Enter the value you want to insert in node 30
-> 30 -> 10 -> 20
Enter the choice of operation you want to do 3
Enter position at which you want to insert data 2
enter the data you want to insert in list 40
-> 30 -> 40 -> 10 -> 20
Enter the choice of operation you want to do 3
Enter position at which you want to insert data 2
enter the data you want to insert in list 50
-> 30 -> 50 -> 40 -> 10 -> 20
Enter the choice of operation you want to do 4
Data deleted from list is 30
-> 50 -> 40 -> 10 -> 20
Enter the choice of operation you want to do 5

Data deleted from list is 20
-> 50 -> 40 -> 10
Enter the choice of operation you want to do 6
Enter the position you want to delete2
data deleted from list is 40
-> 50 -> 10
Enter the choice of operation you want to do 8
```