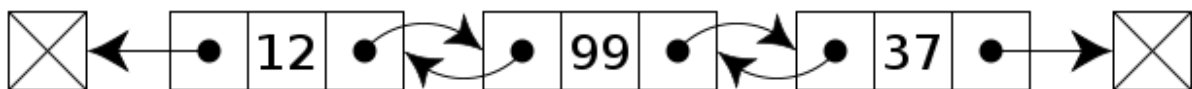


Double Link List

कंप्यूटर विज्ञान में, एक दोगुना से जुड़े सूची नोड्स कहा जाता है क्रमिक रूप से जुड़े हुए अभिलेखों का एक सेट होता है कि किसी लिंक किए गए डेटा संरचना है. प्रत्येक नोड पिछले करने के लिए और नोड्स के क्रम में अगले नोड के लिए जिक्र कर रहे हैं कि लिंक नामक दो क्षेत्रों, शामिल हैं. शुरुआत और सूची की चंक्रमण की सुविधा के लिए टर्मिनेटर के कुछ प्रकार, आम तौर पर एक प्रहरी नोड या अशक्त करने के लिए नोड्स 'पिछले और अगले क्रमशः लिंक,, बिंदु, समाप्त. केवल एक प्रहरी नोड है, तो सूची चक्राकार प्रहरी नोड के माध्यम से जुड़ा हुआ है. यह एक ही डेटा आइटम से गठित दो अकेले लिंक सूचियों रूप में की अवधारणा, लेकिन विपरीत अनुक्रमिक आदेश में किया जा सकता है.



दो नोड लिंक या तो दिशा में सूची की चंक्रमण अनुमति देते हैं. एक दोगुना लिंक की गई सूची में एक नोड जोड़ने या हटाने के एक अकेले लिंक सूची पर एक ही आपरेशन से अधिक लिंक को बदलने की आवश्यकता है जबकि ट्रैक रखने की कोई जरूरत नहीं है, क्योंकि आपरेशन (प्रथम नोड्स के अलावा अन्य नोड्स के लिए) आसान है और संभवतः अधिक कुशल हैं चंक्रमण या उसके लिंक संशोधित किया जा सकता है, ताकि पिछले नोड खोजने के लिए सूची को पार करने की कोई आवश्यकता दौरान पिछले नोड की.

CODES{C}

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
    struct node *previous;
    int data;
    struct node *next;
}*head, *last;

void insert_begning(int value)
{
    struct node *var,*temp;
    var=(struct node *)malloc(sizeof(struct node));
    var->data=value;
    if(head==NULL)
    {
        head=var;
        head->previous=NULL;
        head->next=NULL;
        last=head;
    }
    else
    {
        temp=var;
        temp->previous=NULL;
        temp->next=head;
        head->previous=temp;
        head=temp;
    }
}

void insert_end(int value)
{
    struct node *var,*temp;
    var=(struct node *)malloc(sizeof(struct node));
    var->data=value;
    if(head==NULL)
    {
        head=var;
```

```

        head->previous=NULL;
        head->next=NULL;
        last=head;
    }
    else
    {
        last=head;
        while(last!=NULL)
        {
            temp=last;
            last=last->next;
        }
        last=var;
        temp->next=last;
        last->previous=temp;
        last->next=NULL;
    }
}

int insert_after(int value, int loc)
{
    struct node *temp,*var,*temp1;
    var=(struct node *)malloc(sizeof(struct node));
    var->data=value;
    if(head==NULL)
    {
        head=var;
        head->previous=NULL;
        head->next=NULL;
    }
    else
    {
        temp=head;
        while(temp!=NULL && temp->data!=loc)
        {
            temp=temp->next;
        }
        if(temp==NULL)
        {
            printf("\n%d is not present in list ",loc);
        }
        else
        {

```

```

        temp1=temp->next;
        temp->next=var;
        var->previous=temp;
        var->next=temp1;
        temp1->previous=var;
    }
}
last=head;
while(last->next!=NULL)
{
    last=last->next;
}
}
int delete_from_end()
{
    struct node *temp;
    temp=last;
    if(temp->previous==NULL)
    {
        free(temp);
        head=NULL;
        last=NULL;
        return 0;
    }
    printf("\nData deleted from list is %d \n",last->data);
    last=temp->previous;
    last->next=NULL;
    free(temp);
    return 0;
}

int delete_from_middle(int value)
{
    struct node *temp,*var,*t, *temp1;
    temp=head;
    while(temp!=NULL)
    {
        if(temp->data == value)
        {
            if(temp->previous==NULL)
            {
                free(temp);
                head=NULL;
            }
        }
    }
}

```

```

        last=NULL;
        return 0;
    }
    else
    {
        var->next=temp1;
        temp1->previous=var;
        free(temp);
        return 0;
    }
}
else
{
    var=temp;
    temp=temp->next;
    temp1=temp->next;
}
}
printf("data deleted from list is %d",value);
}

void display()
{
    struct node *temp;
    temp=head;
    if(temp==NULL)
    {
        printf("List is Empty");
    }
    while(temp!=NULL)
    {
        printf("-> %d ",temp->data);
        temp=temp->next;
    }
}

int main()
{
    int value, i, loc;
    head=NULL;
    printf("Select the choice of operation on link list");
    printf("\n1.) insert at begning\n2.) insert at at\n3.) insert at middle");
    printf("\n4.) delete from end\n5.) reverse the link list\n6.) display list\n7.)exit");

```

```
while(1)
{
    printf("\n\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);
            insert_begning(value);
            display();
            break;
        }
        case 2:
        {
            printf("enter the value you want to insert in node at last ");
            scanf("%d",&value);
            insert_end(value);
            display();
            break;
        }
        case 3:
        {
            printf("after which data you want to insert data ");
            scanf("%d",&loc);
            printf("enter the data you want to insert in list ");
            scanf("%d",&value);
            insert_after(value,loc);
            display();
            break;
        }
        case 4:
        {
            delete_from_end();
            display();
            break;
        }
        case 5:
        {
            printf("enter the value you want to delete");
            scanf("%d",value);
            delete_from_middle(value);
        }
    }
}
```

```
        display();
        break;
    }
    case 6 :
    {
        display();
        break;
    }
    case 7 :
    {
        exit(0);
        break;
    }
}
}
printf("\n\n%d",last->data);
display();
getch();
}
```

CODES{JAVA}

```
import java.util.Scanner;
```

```
/* Class Node */  
class Node  
{  
    protected int data;  
    protected Node next, prev;  
  
    /* Constructor */  
    public Node()  
    {  
        next = null;  
        prev = null;  
        data = 0;  
    }  
    /* Constructor */  
    public Node(int d, Node n, Node p)  
    {  
        data = d;  
        next = n;  
        prev = p;  
    }  
    /* Function to set link to next node */  
    public void setLinkNext(Node n)  
    {  
        next = n;  
    }  
    /* Function to set link to previous node */  
    public void setLinkPrev(Node p)  
    {  
        prev = p;  
    }  
}
```



```

/* Funtion to get link to next node */
public Node getLinkNext()
{
    return next;
}

/* Function to get link to previous node */
public Node getLinkPrev()
{
    return prev;
}

/* Function to set data to node */
public void setData(int d)
{
    data = d;
}

/* Function to get data from node */
public int getData()
{
    return data;
}
}

/* Class linkedList */
class linkedList
{
    protected Node start;
    protected Node end ;
    public int size;

    /* Constructor */

```

```
public linkedList()
{
    start = null;
    end = null;
    size = 0;
}
/* Function to check if list is empty */
public boolean isEmpty()
{
    return start == null;
}
/* Function to get size of list */
public int getSize()
{
    return size;
}
/* Function to insert element at begining */
public void insertAtStart(int val)
{
    Node nptr = new Node(val, null, null);
    if(start == null)
    {
        start = nptr;
        end = start;
    }
    else
    {
        start.setLinkPrev(nptr);
        nptr.setLinkNext(start);
        start = nptr;
    }
}
```

```

    }
    size++;
}

/* Function to insert element at end */
public void insertAtEnd(int val)
{
    Node nptr = new Node(val, null, null);
    if(start == null)
    {
        start = nptr;
        end = start;
    }
    else
    {
        nptr.setLinkPrev(end);
        end.setLinkNext(nptr);
        end = nptr;
    }
    size++;
}

/* Function to insert element at position */
public void insertAtPos(int val , int pos)
{
    Node nptr = new Node(val, null, null);
    if (pos == 1)
    {
        insertAtStart(val);
        return;
    }
    Node ptr = start;

```

```

for (int i = 2; i <= size; i++)
{
    if (i == pos)
    {
        Node tmp = ptr.getLinkNext();
        ptr.setLinkNext(nptr);
        nptr.setLinkPrev(ptr);
        nptr.setLinkNext(tmp);
        tmp.setLinkPrev(nptr);
    }
    ptr = ptr.getLinkNext();
}
size++ ;
}

/* Function to delete node at position */
public void deleteAtPos(int pos)
{
    if (pos == 1)
    {
        if (size == 1)
        {
            start = null;
            end = null;
            size = 0;
            return;
        }
        start = start.getLinkNext();
        start.setLinkPrev(null);
        size--;
        return ;
    }
}

```

```

    }
    if (pos == size)
    {
        end = end.getLinkPrev();
        end.setLinkNext(null);
        size-- ;
    }
    Node ptr = start.getLinkNext();
    for (int i = 2; i <= size; i++)
    {
        if (i == pos)
        {
            Node p = ptr.getLinkPrev();
            Node n = ptr.getLinkNext();

            p.setLinkNext(n);
            n.setLinkPrev(p);
            size-- ;
            return;
        }
        ptr = ptr.getLinkNext();
    }
}

/* Function to display status of list */
public void display()
{
    System.out.print("\nDoubly Linked List = ");
    if (size == 0)
    {
        System.out.print("empty\n");
    }
}

```

```

        return;
    }
    if (start.getLinkNext() == null)
    {
        System.out.println(start.getData() );
        return;
    }
    Node ptr = start;
    System.out.print(start.getData()+ " <-> ");
    ptr = start.getLinkNext();
    while (ptr.getLinkNext() != null)
    {
        System.out.print(ptr.getData()+ " <-> ");
        ptr = ptr.getLinkNext();
    }
    System.out.print(ptr.getData()+ "\n");
}

}

/* Class DoublyLinkedList */
public class DoublyLinkedList
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        /* Creating object of linkedList */
        linkedList list = new linkedList();
        System.out.println("Doubly Linked List Test\n");
        char ch;
        /* Perform list operations */

```

```

do
{
    System.out.println("\nDoubly Linked List Operations\n");
    System.out.println("1. insert at begining");
    System.out.println("2. insert at end");
    System.out.println("3. insert at position");
    System.out.println("4. delete at position");
    System.out.println("5. check empty");
    System.out.println("6. get size");

    int choice = scan.nextInt();
    switch (choice)
    {
    case 1 :
        System.out.println("Enter integer element to insert");
        list.insertAtStart( scan.nextInt() );
        break;
    case 2 :
        System.out.println("Enter integer element to insert");
        list.insertAtEnd( scan.nextInt() );
        break;
    case 3 :
        System.out.println("Enter integer element to insert");
        int num = scan.nextInt() ;
        System.out.println("Enter position");
        int pos = scan.nextInt() ;
        if (pos < 1 || pos > list.getSize() )
            System.out.println("Invalid position\n");
        else
            list.insertAtPos(num, pos);
    }
}

```

```

        break;
    case 4 :
        System.out.println("Enter position");
        int p = scan.nextInt() ;
        if (p < 1 || p > list.getSize() )
            System.out.println("Invalid position\n");
        else
            list.deleteAtPos(p);
        break;
    case 5 :
        System.out.println("Empty status = "+ list.isEmpty());
        break;
    case 6 :
        System.out.println("Size = "+ list.getSize() +" \n");
        break;
    default :
        System.out.println("Wrong Entry \n ");
        break;
    }
    /* Display List */
    list.display();
    System.out.println("\nDo you want to continue (Type y or n) \n");
    ch = scan.next().charAt(0);

    } while (ch == 'Y' || ch == 'y');
}
}

```