**PROGRAMS:** **Implementation of Doubly Linked List**

**Input:**

#include<stdio.h>

#include<stdlib.h>

struct node

{

    int data;

    struct node \*prev;

    struct node \*next;

};

struct node \*start=NULL;

void insert\_beg()

{

    int x;

    printf("Enter the element to be inserted:");

    scanf("%d",&x);

    struct node \*tmp;

    tmp=(struct node\*)malloc(sizeof(struct node));

    if(start==NULL)

    {

        tmp->data=x;

        tmp->prev=NULL;

        tmp->next=NULL;

        start=tmp;

        printf("Insertion of %d done\n",x);

    }

    else

    {

        tmp->data=x;

        tmp->next=start;

        tmp->prev=NULL;

        start->prev=tmp;

        start=tmp;

        printf("Insertion of %d done\n",x);

        return;

    }

}

Expt no: 6

void insert\_end()

{

    int x;

    printf("Enter the element to be inserted:");

    scanf("%d",&x);

    struct node \*tmp,\*p;

    tmp=(struct node\*)malloc(sizeof(struct node));

    if(start==NULL)

    {

        tmp->data=x;

        tmp->prev=NULL;

        tmp->next=NULL;

        start=tmp;

        printf("Insertion of %d done\n",x);

    }

    else

    {

        p=start;

        while(p->next!=NULL)

        {p=p->next;}

    }

    tmp->data=x;

    tmp->prev=p;

    tmp->next=NULL;

    p->next=tmp;

    printf("Insertion of %d done\n",x);

}

void insert\_afternode()

{

    int x,item;

    struct node \*tmp,\*p;

    printf("Enter the element to be inserted:");

    scanf("%d",&x);

    printf("Enter the element after which it is to be inserted:");

    scanf("%d",&item);

    p=start;

    while(p!=NULL)

    {

        if(p->data==item)

        {

                tmp=(struct node\*)malloc(sizeof(struct node));

                tmp->data=x;

                tmp->next=p->next;

                if(p->next!=NULL)

                {p->next->prev=tmp;}

                p->next=tmp;

                tmp->prev=p;

                printf("Insertion of %d after %d done\n",x,item);

                return start;

        }

        p=p->next;

    }

    printf("%d not present in the list\n",item);

    return start;

}

void insert\_beforenode()

{

    int x,item;

    struct node \*tmp,\*p;

    printf("Enter the element to be inserted:");

    scanf("%d",&x);

    printf("Enter the element before which it is to be inserted:");

    scanf("%d",&item);

    if(start==NULL)

    {

        printf("Empty list\n");

        return start;

    }

    if(start->data==item)

    {

Expt no: 6

            tmp=(struct node\*)malloc(sizeof(struct node));

            tmp->data=x;

            tmp->next=start;

            tmp->prev=NULL;

            start=tmp;

            printf("Insertion of %d before %d done\n",x,item);

            return start;

    }

    p=start;

    while(p->next!=NULL)

    {

        if(p->next->data==item)

        {

            tmp=(struct node\*)malloc(sizeof(struct node));

            tmp->data=x;

            tmp->next=p->next;

            p->next->prev=tmp;

            p->next=tmp;

            tmp->prev=p;

            printf("Insertion of %d before %d done\n",x,item);

            return start;

        }

        p=p->next;

    }

    printf("%d not present in the list\n",item);

    return start;

}

void insert\_atpos()

{

    struct node \*p,\*temp;

    int pos,i,x;

    printf("Enter element to be inserted:");

    scanf("%d",&x);

    printf("Enter position to be inserted at:");

    scanf("%d",&pos);

    temp=(struct node \*)malloc(sizeof(struct node));

    temp->data=x;

    if(pos==1)

    {

        temp->next=start;

        temp->prev=NULL;

        start->prev=temp;

        start=temp;

        return;

    }

        p=start;

        for(i=1; i<pos-1 && p!=NULL;i++)

        {p=p->next;}

        if(p==NULL)

        {

            printf("Less than %d elements present\n",pos);

            return;

        }

        else

        {

            temp->next=p->next;

            temp->prev=p;

            if(p->next!=NULL)

                p->next->prev=temp;

            p->next=temp;

        }

        return;

}

void delete\_beg()

{

    struct node \*p,\*temp;

    if(start==NULL)

    {

        printf("List is empty\n");

        return;

    }

    if(start->next==NULL)

Expt no: 6

    {

        printf("Element %d deleted\n",start->data);

        p=start;

        start=NULL;

        free(p);

        return;

    }

    temp=start;

    start->next->prev=NULL;

    start=start->next;

    printf("Element %d deleted\n",temp->data);

    free(temp);

    return;

}

void delete\_end()

{

    struct node \*p,\*temp;

    if(start==NULL)

    {

        printf("List is empty\n");

        return;

    }

    if(start->next==NULL)

    {

        printf("Element %d deleted\n",start->data);

        start=NULL;

        return;

    }

    p=start;

    while(p->next->next!=NULL)

        {p=p->next;}

    temp=p->next;

    p->next=NULL;

    printf("Element %d deleted\n",temp->data);

    free(temp);

    return;

}

void delete\_betw()

{

    struct node \*p,\*temp;

    int element;

    if(start==NULL)

    {

        printf("List is empty\n");

        return;

    }

    printf("Enter element to be deleted:");

    scanf("%d",&element);

    if(start->data==element)

    {

        temp=start;

        if(start->next!=NULL)

        start->next->prev=NULL;

        start=start->next;

        free(temp);

        return;

    }

    p=start;

    while(p->data!=element)

        p=p->next;

    temp=p;

    p->prev->next=p->next;

    if(p->next!=NULL)

        p->next->prev=p->prev;

    printf("Element %d deleted\n",temp->data);

    free(temp);

    return;

}

void delete\_atpos()

{

    struct node \*p,\*temp;

    int pos,i;

    temp=(struct node \*)malloc(sizeof(struct node));

    printf("Enter position to be deleted:");

Expt no: 6

    scanf("%d",&pos);

    if(pos==1)

    {

        temp=start;

        if(start->next!=NULL)

            start->next->prev=NULL;

        start=start->next;

        printf("Element %d deleted\n",temp->data);

        free(temp);

            return;

    }

    else

    {

        p=start;

        for(i=1; i<pos && p!=NULL;i++)

            p=p->next;

        if(p==NULL)

        {

            printf("%d not present in the list\n",pos);

            return;

        }

        else

        {

            temp=p;

            p->prev->next=p->next;

            if(p->next!=NULL)

                p->next->prev=p->prev;

            printf("Element %d deleted\n",temp->data);

            free(temp);

        }

        return;

    }

}

void display()

{

    struct node \*p;

    p=start;

    if(p==NULL)

    {

        printf("List is empty\n");

        return;

    }

    while(p!=NULL)

    {

        printf("%d ",p->data);

        p=p->next;

    }

    printf("\n");

}

void count()

{

    struct node \*p;

    int count=0;

    p=start;

    while(p!=NULL)

    {

        p=p->next;

        count++;

    }

    printf("count = %d\n",count);

}

void search()

{

    struct node \*p;

    int element,pos=1;

    if(start==NULL)

    {

        printf("List is empty\n");

        return;

    }

    printf("Enter element to be searched:");

    scanf("%d",&element);

    if(start->data==element)

    {

        printf("element %d found at position %d\n",element,pos);

Expt no: 6

        return;

    }

    p=start;

    while(p!=NULL)

    {

        if(p->data==element)

        {

            printf("element %d found at position %d\n",element,pos);

            return;

        }

        p=p->next;

        pos++;

    }

    printf("element not found\n");

}

void reversal()

{

        if(start==NULL)

        {

            printf("List is empty\n");

            return;

        }

        struct node \*p1,\*p2;

        p1=start;

        p2=p1->next;

        p1->next=NULL;

        p1->prev=p2;

        while(p2!=NULL)

        {

            p2->prev=p2->next;

            p2->next=p1;

            p1=p2;

            p2=p2->prev;

        }

        start=p1;

        printf("reversal completed\n");

        return;

}

int main()

{

    int choice;

    printf("1.Insert at the beginning\t2.Insert at the end\n3.Insert after a node\t\t4.Insert before a node\n5.Insert at a given position\t6.Delete at the beginning\n7.Delete at the end\t\t8.Delete in between the list\n9.Delete at a given position\t10.Display\n11.Count\t\t\t12.Search\n13.Reversal\t\t\t14.Exit");

    while(1)

    {

        printf("\nEnter choice :");

        scanf("%d",&choice);

        switch(choice)

        {

            case 1:insert\_beg();

                    break;

            case 2:insert\_end();

                    break;

            case 3:insert\_afternode();

                    break;

            case 4:insert\_beforenode();

                    break;

            case 5:insert\_atpos();

                    break;

            case 6:delete\_beg();

                    break;

            case 7:delete\_end();

                    break;

            case 8:delete\_betw();

                    break;

            case 9:delete\_atpos();

Expt no: 6

                    break;

            case 10:display();

                    break;

            case 11:count();

                    break;

            case 12:search();

                    break;

            case 13:reversal();

                    break;

            case 14:exit(0);

            default:printf("invalid choice\n");

        }

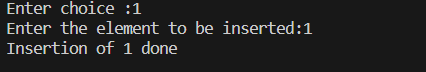
    }

    return 0;

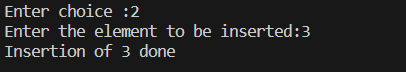
}

**Output:**

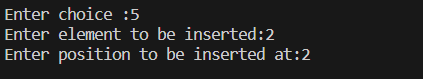
1. Insert at the beginning:



1. Insert at the end:



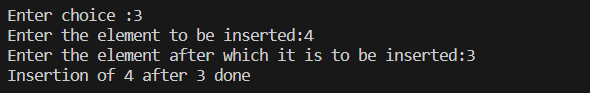
1. Insert at a given position:



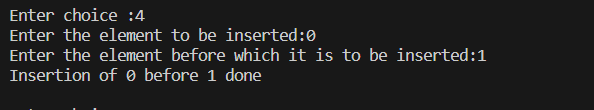
1. Display:



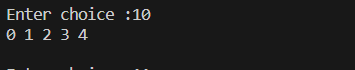
1. Insert after a node:



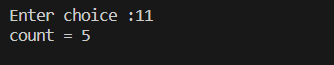
1. Insert before a node:



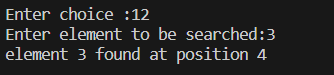
1. Display:



1. Count:

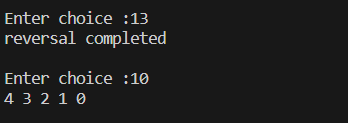


1. Search:



1. Reversal & display:

Expt no: 6



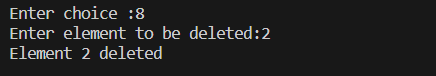
1. Delete at the beginning:



1. Delete at the end:



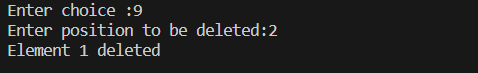
1. Delete in between in the nodes:



1. Display:



1. Delete at a given position:



1. Display:

