SANGOLA COLLEGE, SANGOLA
Class-B.Sc(ECS)-II, SEM-III 2024-25
Practical Assignments
Sub- Data Structure using C++

Assignment No- 4

1) Write a menu driven program to implement singly linear linked list with its different operations.

```
#include<iostream.h>
#include<conio.h>

class node
{
        int info;
        node *next;
        public:
                node* create();

                void ins_beg(int);
                void ins_end(int);
                void ins_bet(int,int);

                int rem_beg();
                int rem_end();
                int rem_bet(int);

                void display();
                void count();
                void reverse();
                void search(int);
}*list;

node* node :: create()
{
        node *z=new node;
```

```
        return(z);
}

void node :: ins_beg(int x)
{
        node *p, *q;
        p=list;
        if(p==NULL)
        {
             p=create();
             p->info=x;
             p->next=NULL;
             list=p;
        }
        else
        {
                q=create();
                q->info=x;
                q->next=p;
                list=q;
        }
        cout<<"Node is inserted...\n";
}

void node :: ins_end(int x)
{
        node *p, *q;
        p=list;
        if(p==NULL)
        {
                p=create();
                p->info=x;
                p->next=NULL;
                list=p;
        }
        else
        {
                while(p->next!=NULL)
                {
```

```
                p=p->next;
            }
            q=create();
            q->info=x;
            q->next=NULL;
            p->next=q;
        }
        cout<<"Node is inserted...\n";
}

void node :: ins_bet(int after, int x)
{
        node *q,*p;
        p=list;
        if(p==NULL || p->next==NULL)
        {
            cout<<"Insert between not possible...\n";
        }
        else
        {
            while(p->next!=NULL)
            {
                if(p->info==after)
                {
                    q=create();
                    q->info=x;
                    q->next=p->next;
                    p->next=q;
                }
                p=p->next;
            }
        }
        cout<<"Node is inserted...\n";
}

int node :: rem_beg()
{
        int z;
        node *p;
```

```
        p=list;
        if(p==NULL)
        {
                cout<<"List is empty...\n";
                return(0);
        }
        else if(p->next==NULL)
        {
                z=p->info;
                delete(p);
                list=NULL;
                return(z);
        }
        else
        {
                z=p->info;
                list=p->next;
                delete(p);
                return(z);
        }
}

int node :: rem_end()
{
        int z;
        node *p, *temp;
        p=list;
        if(p==NULL)
        {
                cout<<"List is empty...\n";
                return(0);
        }
        else if(p->next==NULL)
        {
                z=p->info;
                delete(p);
                list=NULL;
                return(z);
        }
```

```
        else
        {
                while(p->next->next!=NULL)
                {
                        p=p->next;
                }
                temp=p->next;
                z=temp->info;
                p->next=NULL;
                delete(temp);
                return(z);
        }
}

int node :: rem_bet(int after)
{
        int z;
        node *p, *temp;
        p=list;
        if(p==NULL)
        {
                cout<<"List is empty...\n";
                return(0);
        }
        else if(p->next==NULL || p->next->next==NULL)
        {
                cout<<"Remove between not possible...\n";
                return(0);
        }
        else
        {
                while(p->next!=NULL)
                {
                        if(p->info==after)
                        {
                                temp=p->next;
                                z=temp->info;
                                p->next=temp->next;
                                delete(temp);
```

```
                        return(z);
                    }
                    p=p->next;
                }
                return(0);
        }
}

void node :: display()
{
        node *p;
        p=list;
        cout<<"Nodes : \n";
        while(p!=NULL)
        {
                cout<<"\t"<<p->info;
                p=p->next;
        }
}

void node :: count()
{
        int c=0;
        node *p=list;
        while(p!=NULL)
        {
                c++;
                p=p->next;
        }
        cout<<"Total nodes : "<<c<<"\n";
}

void node :: reverse()
{
        node *t1, *t2, *t3=NULL;
        t1=list;
        while(t1!=NULL)
        {
                t2=t1->next;
```

```
            t1->next=t3;
            t3=t1;
            t1=t2;
        }
        list=t3;
        cout<<"Node revesed...\n";
}

void node :: search(int x)
{
        int c=0;
        node *p=list;
        while(p!=NULL)
        {
            if(p->info==x)
            {
                c=1;
                break;
            }
            p=p->next;
        }
        if(c==1)
        {
            cout<<"Node is found...\n";
        }
        else
        {
            cout<<"Node is not found...\n";
        }
}

void main()
{
        int af, n, ch, z;
        clrscr();
        node obj;
        do
        {
            cout<<"\nEnter your choise : \n";
```

```
        cout<<"\t1.Insert Begin.\n \t2.Insert End\n \t3.Insert
Between\n \t4.Remove Begin\n \t5.Remove End\n \t6.Remove
Between\n \t7.Display\n \t8.Count\n \t9.Reverse\n \t10.Search\n
\t11.Exit\n \t";
        cin>>ch;

        switch(ch)
        {
            case 1:
                cout<<"Enter value : ";
                cin>>n;
                obj.ins_beg(n);
                break;

            case 2:
                cout<<"Enter value : ";
                cin>>n;
                obj.ins_end(n);
                break;

            case 3:
                cout<<"After which value : ";
                cin>>af;
                cout<<"Enter value : ";
                cin>>n;
                obj.ins_bet(af,n);
                break;

            case 4:
                z=obj.rem_beg();
                cout<<"Removed value : "<<z;
                break;

            case 5:
                z=obj.rem_end();
                cout<<"Removed value : "<<z;
                break;

            case 6:
```

```
                cout<<"After which value : ";
                cin>>af;
                z=obj.rem_bet(af);
                cout<<"Removed value : "<<z;
                break;

            case 7:
                obj.display();
                break;

            case 8:
                obj.count();
                break;

            case 9:
                obj.reverse();
                break;

            case 10:
                cout<<"Search node : ";
                cin>>n;
                obj.search(n);
                break;

            case 11:
                cout<<"Exit program...";
                break;

            default:
                cout<<"Wrong input...";
                break;
        }
    }while(ch!=11);

    getch();
}
```

2) Write a menu program to implement stack by using linked list.(Dynamic implementation of stack)

```
#include<iostream.h>
#include<conio.h>

class node
{
        int info;
        node *next;
        public:
                node* create();
                void isempty();
                void push(int);
                int  pop();
                void display();
}*list;

node* node :: create()
{
         node *z= new node;
         return(z);
}

void node :: isempty()
{
        if(list==NULL)
        {
                cout<<"Stack is empty...\n";
        }
        else
        {
                cout<<"Stack is NOT empty...\n";
        }
}

void node :: push(int x)
{
        node *p=list, *q;
```

```
        if(p==NULL)
        {
                p=create();
                p->info=x;
                p->next=NULL;
                list=p;
        }
        else
        {
                q=create();
                q->info=x;
                q->next=p;
                list=q;
        }
}

int node :: pop()
{
        int z;
        node *p=list;
        if(p==NULL)
        {
                cout<<"Stack underflows...\n";
                return (0);
        }
        else if(p->next==NULL)
        {
                z=p->info;
                delete(p);
                list=NULL;
                return(z);
        }
        else
        {
                z=p->info;
                list=p->next;
                delete(p);
                return(z);
        }
```

```
}

void node :: display()
{
        node *p=list;
        cout<<"Stack element :\n";
        while(p!=NULL)
        {
                cout<<"\t"<<p->info;
                p=p->next;
        }
        cout<<"\n";
}

void main()
{
        int ch, x;
        node obj;
        clrscr();

        do
        {
                cout<<"\nEnter your choice :\n";
                cout<<"1. Isempty\n2. Push\n3. Pop\n4. Display\n5.
Exit\n\t";
                cin>>ch;
                switch(ch)
                {
                        case 1:
                                obj.isempty();
                                break;

                        case 2:
                                cout<<"Enter element : ";
                                cin>>x;
                                obj.push(x);
                                break;

                        case 3:
```

```
                        x=obj.pop();
                        cout<<"Poped element : "<<x<<"\n";
                        break;

                case 4:
                        obj.display();
                        break;

                case 5:
                        cout<<"Program stoped...";
                        break;
                }
        }while(ch!=5);
        getch();
}
```

3) Write a menu program to implement queue by using linked list.(Dynamic implementation of queue)

```
#include<iostream.h>
#include<conio.h>

class queue
{
        int info;
        queue *next;
        public:
                queue* create();
                void isempty();
                void insert(int);
                int  remove();
                void display();
}*list;

queue* queue :: create()
{
        queue *z=new queue;
```

```
        return(z);
}

void queue :: isempty()
{
   queue *q=list;
   if(q==NULL)
   {
        cout<<"Queue is empty...\n";
   }
   else
   {
        cout<<"Queue is NOT empty...\n";
   }
}

void queue :: insert(int x)
{
   queue *q=list, *p;
   if(q==NULL)
   {
        q=create();
        q->info=x;
        q->next=NULL;
        list=q;
   }
   else
   {
        while(q->next!=NULL)
        {
          q=q->next;
        }
        p=create();
        p->info=x;
        p->next=NULL;
        q->next=p;
   }
   cout<<"Element inserted...\n";
}
```

```
int queue :: remove()
{
        int z;
        queue *q=list;
        if(q==NULL)
        {
                cout<<"Queue Underflows...\n";
                return(0);
        }
        else if(q->next==NULL)
        {
                z=q->info;
                delete(q);
                list=NULL;
                return(z);
        }
        else
        {
                z=q->info;
                list=q->next;
                delete(q);
                return(z);
        }
}

void queue :: display()
{
        queue *q=list;
        cout<<"Elements : \n";
        while(q!=NULL)
        {
                cout<<"\t"<<q->info;
                q=q->next;
        }
        cout<<"\n";
}

void main()
```

```
{
	int ch, x;
	queue obj;

	clrscr();

	do
	{
		cout<<"\nEnter your choise : ";
		cout<<"\n1. Isempty\n2. Insert\n3. Remove\n4. Display\n5.
Exit\n\t";
		cin>>ch;

		switch(ch)
		{
			case 1:
				obj.isempty();
				break;

			case 2:
				cout<<"Enter value : ";
				cin>>x;
				obj.insert(x);
				break;

			case 3:
				x=obj.remove();
				cout<<"Removed value : "<<x<<"\n";
				break;

			case 4:
				obj.display();
				break;

			case 5:
				cout<<"Exit program...\n";
				break;

			default:
```

```
                        cout<<"Wrong input...\n";
                        break;
                }
        }while(ch!=5);
getch();
}
```

4) Write a menu program to implement doubly linear linked list with its different operations.

```
#include<iostream.h>
#include<conio.h>

class node
{
        int info;
        node *next, *prev;
        public:
                node* create();

                void ins_beg(int);
                void ins_end(int);
                void ins_bet(int,int);

                int rem_beg();
                int rem_end();
                int rem_bet(int);

                void display();
                void count();
                void reverse();
                void search(int);
}*list;

node* node :: create()
{
        node *z=new node;
```

```
        return(z);
}

void node :: ins_beg(int x)
{
        node *p, *q;
        p=list;
        if(p==NULL)
        {
             p=create();
             p->prev=NULL;
             p->info=x;
             p->next=NULL;
             list=p;
        }
        else
        {
                q=create();
                q->prev=NULL;
                q->info=x;
                q->next=p;
                p->prev=q;
                list=q;
        }
        cout<<"Node is inserted...\n";
}

void node :: ins_end(int x)
{
        node *p, *q;
        p=list;
        if(p==NULL)
        {
                p=create();
                p->prev=NULL;
                p->info=x;
                p->next=NULL;
                list=p;
        }
```

```
        else
        {
                while(p->next!=NULL)
                {
                        p=p->next;
                }
                q=create();
                q->prev=p;
                q->info=x;
                q->next=NULL;
                p->next=q;
        }
        cout<<"Node is inserted...\n";
}

void node :: ins_bet(int after, int x)
{
        node *q,*p;
        p=list;
        if(p==NULL || (p->prev==NULL && p->next==NULL))
        {
                cout<<"Insert between not possible...\n";
        }
        else
        {
                while(p->next!=NULL)
                {
                        if(p->info==after)
                        {
                                q=create();
                                q->prev=p;
                                q->info=x;
                                q->next=p->next;
                                p->next->prev=q;
                                p->next=q;
                        }
                        p=p->next;
                }
        }
```

```
        cout<<"Node is inserted...\n";
}

int node :: rem_beg()
{
        int z;
        node *p;
        p=list;
        if(p==NULL)
        {
                cout<<"List is empty...\n";
                return(0);
        }
        else if(p->prev==NULL && p->next==NULL)
        {
                z=p->info;
                delete(p);
                list=NULL;
                return(z);
        }
        else
        {
                z=p->info;
                p->next->prev=NULL;
                list=p->next;
                delete(p);
                return(z);
        }
}

int node :: rem_end()
{
        int z;
        node *p, *temp;
        p=list;
        if(p==NULL)
        {
                cout<<"List is empty...\n";
                return(0);
```

```
        }
        else if(p->prev==NULL && p->next==NULL)
        {
                z=p->info;
                delete(p);
                list=NULL;
                return(z);
        }
        else
        {
                while(p->next->next!=NULL)
                {
                        p=p->next;
                }
                temp=p->next;
                z=temp->info;
                p->next=NULL;
                delete(temp);
                return(z);
        }
}

int node :: rem_bet(int after)
{
        int z;
        node *p, *temp;
        p=list;
        if(p==NULL)
        {
                cout<<"List is empty...\n";
                return(0);
        }
        else if((p->prev==NULL && p->next==NULL) || (p->prev==NULL
&& p->next->next==NULL))
        {
                cout<<"Remove between not possible...\n";
                return(0);
        }
        else
```

```
        {
                while(p->next!=NULL)
                {
                        if(p->info==after)
                        {
                                temp=p->next;
                                z=temp->info;
                                temp->next->prev=p;
                                p->next=temp->next;
                                delete(temp);
                                return(z);
                        }
                        p=p->next;
                }
                return(0);
        }
}

void node :: display()
{
        node *p;
        p=list;
        cout<<"Nodes : \n";
        while(p!=NULL)
        {
                cout<<"\t"<<p->info;
                p=p->next;
        }
}

void node :: count()
{
        int c=0;
        node *p=list;
        while(p!=NULL)
        {
                c++;
                p=p->next;
        }
```

```
        cout<<"Total nodes : "<<c<<"\n";
}

void node :: reverse()
{
        node *t1, *t2, *t3=NULL;
        t1=list;
        while(t1!=NULL)
        {
                t2=t1->next;
                t1->next=t3;
                t1->prev=t2;
                t3=t1;
                t1=t2;
        }
        list=t3;
        cout<<"Node revesed...\n";
}

void node :: search(int x)
{
        int c=0;
        node *p=list;
        while(p!=NULL)
        {
                if(p->info==x)
                {
                        c=1;
                        break;
                }
                p=p->next;
        }
        if(c==1)
        {
                cout<<"Node is found...\n";
        }
        else
        {
                cout<<"Node is not found...\n";
```

```
        }
}

void main()
{
        int af, n, ch, z;
        clrscr();
        node obj;
        do
        {
                cout<<"\nEnter your choise : \n";
                cout<<"\t1.Insert Begin.\n \t2.Insert End\n \t3.Insert
Between\n \t4.Remove Begin\n \t5.Remove End\n \t6.Remove
Between\n \t7.Display\n \t8.Count\n \t9.Reverse\n \t10.Search\n
\t11.Exit\n \t";
                cin>>ch;

                switch(ch)
                {
                        case 1:
                                cout<<"Enter value : ";
                                cin>>n;
                                obj.ins_beg(n);
                                break;

                        case 2:
                                cout<<"Enter value : ";
                                cin>>n;
                                obj.ins_end(n);
                                break;

                        case 3:
                                cout<<"After which value : ";
                                cin>>af;
                                cout<<"Enter value : ";
                                cin>>n;
                                obj.ins_bet(af,n);
                                break;
```

```
case 4:
        z=obj.rem_beg();
        cout<<"Removed value : "<<z;
        break;

case 5:
        z=obj.rem_end();
        cout<<"Removed value : "<<z;
        break;

case 6:
        cout<<"After which value : ";
        cin>>af;
        z=obj.rem_bet(af);
        cout<<"Removed value : "<<z;
        break;

case 7:
        obj.display();
        break;

case 8:
        obj.count();
        break;

case 9:
        obj.reverse();
        break;

case 10:
        cout<<"Search node : ";
        cin>>n;
        obj.search(n);
        break;

case 11:
        cout<<"Exit program...";
        break;
```

```
                default:
                        cout<<"Wrong input...";
                        break;
                }

        }while(ch!=11);

        getch();
}
```

5) Write a menu program to implement singly circular linked list with its different operations.

```
#include<iostream.h>
#include<conio.h>

class node
{
        int info;
        node *next;
        public:
                node* create();
                void ins_beg(int);
                void ins_end(int);
                void ins_bet(int, int);

                int rem_beg();
                int rem_end();
                int rem_bet(int);

                void display();
                void count();
                void reverse();
                void search(int);
}*list;

node* node :: create()
```

```
{
        node *z= new node;
        return(z);
}

void node :: ins_beg(int x)
{
        node *p=list, *q;
        if(p==NULL)
        {
                p=create();
                p->info=x;
                p->next=p;
                list=p;
        }
        else
        {
                while(p->next!=list)
                {
                        p=p->next;
                }
                q=create();
                q->info=x;
                q->next=p->next;
                p->next=q;
                list=q;
        }
        cout<<"Insert successfully...\n";
}

void node :: ins_end(int x)
{
        node *p=list, *q;
        if(p==NULL)
        {
                p=create();
                p->info=x;
                p->next=p;
                list=p;
```

```
        }
        else
        {
                while(p->next!=list)
                {
                        p=p->next;
                }
                q=create();
                q->info=x;
                q->next=p->next;
                p->next=q;
        }
        cout<<"Insert successfully...\n";
}

void node :: ins_bet(int aft, int x)
{
        node *p=list, *q;
        if(p==NULL || p->next==p)
        {
                cout<<"Insert between NOT possible...\n";
        }
        else
        {
                while(p->next!=list)
                {
                        if(p->info==aft)
                        {
                                q=create();
                                q->info=x;
                                q->next=p->next;
                                p->next=q;
                        }
                        p=p->next;
                }
        }
        cout<<"Insert successfully...\n";
}
```

```
int node :: rem_beg()
{
	int z;
	node *p=list, *temp;
	if(p==NULL)
	{
		cout<<"List is empty...\n";
		return(0);
	}
	else if(p->next==p)
	{
		z=p->info;
		delete(p);
		list=NULL;
		return(z);
	}
	else
	{
		while(p->next!=list)
		{
			p=p->next;
		}
		temp=p->next;
		z=temp->info;
		p->next=temp->next;
		list=temp->next;
		delete(temp);
		return(z);
	}
	cout<<"Remove successfully...\n";
}

int node :: rem_end()
{
	int z;
	node *p=list, *temp;
	if(p==NULL)
	{
		cout<<"List is empty...\n";
```

```
            return(0);
        }
        else if(p->next==p)
        {
            z=p->info;
            delete(p);
            list=NULL;
            return(z);
        }
        else
        {
            while(p->next->next!=p)
            {
                p=p->next;
            }
            temp=p->next;
            z=temp->info;
            p->next=temp->next;
            delete(temp);
            return(z);
        }
        cout<<"Remove successfully...\n";
}

int node :: rem_bet(int aft)
{
        int z;
        node *p=list, *temp;
        if(p==NULL)
        {
            cout<<"List is empty...\n";
            return(0);
        }
        else if(p->next==p || p->next->next==p)
        {
            cout<<"Remove between NOT possible...\n";
            return(0);
        }
        else
```

```
	{
		while(p->next!=list)
		{
			if(p->info==aft)
			{
				temp=p->next;
				z=temp->info;
				p->next=temp->next;
				delete(temp);
				return(z);
			}
			p=p->next;
		}
		return(0);
	}
	cout<<"Remove successfully...\n";
}

void node :: display()
{
	node *p=list;
	cout<<"Nodes in list : \n";
	do
	{
		cout<<"\t"<<p->info;
		p=p->next;
	}while(p!=list);
}

void node :: count()
{
	int c=0;
	node *p=list;
	do
	{
		c++;
		p=p->next;
	}while(p!=list);
	cout<<"Total nodes in list is : \n"<<c;
```

```
}

void node :: reverse()
{
        node *t1=list, *t2, *t3=list;
        do
        {
                t2=t1->next;
                t1->next=t3;
                t3=t1;
                t1=t2;
        }while(t1!=list);
        list=t3;
        t1->next=t3;
        cout<<"List is reversed...\n";
}

void node :: search(int x)
{
        int t=0;
        node *p=list;
        do
        {
                if(p->info==x)
                {
                        t=1;
                        break;
                }
                p=p->next;

        }while(p!=list);

        if(t==1)
        {
                cout<<"Node is found...\n";
        }
        else
        {
                cout<<"Node is not found...\n";
```

```
        }
}

void main()
{
        int af, n, ch, z;
        clrscr();
        node obj;
        do
        {
                cout<<"\nEnter your choise : ";
                cout<<"\n \t1.Insert Begin.\n \t2.Insert End\n \t3.Insert
Between\n \t4.Remove Begin\n \t5.Remove End\n \t6.Remove
Between\n \t7.Display\n \t8.Count\n \t9.Reverse\n \t10.Search\n
\t11.Exit\n \t";
                cin>>ch;

                switch(ch)
                {
                        case 1:
                                cout<<"Enter value : ";
                                cin>>n;
                                obj.ins_beg(n);
                                break;

                        case 2:
                                cout<<"Enter value : ";
                                cin>>n;
                                obj.ins_end(n);
                                break;

                        case 3:
                                cout<<"After which value : ";
                                cin>>af;
                                cout<<"Enter value : ";
                                cin>>n;
                                obj.ins_bet(af,n);
                                break;
```

```
case 4:
        z=obj.rem_beg();
        cout<<"Removed value : "<<z<<"\n";
        break;

case 5:
        z=obj.rem_end();
        cout<<"Removed value : "<<z<<"\n";
        break;

case 6:
        cout<<"After which value : ";
        cin>>af;
        z=obj.rem_bet(af);
        cout<<"Removed value : "<<z<<"\n";
        break;

case 7:
        obj.display();
        break;

case 8:
        obj.count();
        break;

case 9:
        obj.reverse();
        break;

case 10:
        cout<<"Search node : ";
        cin>>n;
        obj.search(n);
        break;

case 11:
        cout<<"Exit program...";
        break;
```

```
                default:
                        cout<<"Wrong input...";
                        break;
                }
        }while(ch!=11);

        getch();
}
```

6) Write a menu driven program to implement doubly circular linked list with its different operations.

```
#include<iostream.h>
#include<conio.h>

class node
{
        int info;
        node *prev, *next;
        public:
                node* create();
                void ins_beg(int);
                void ins_end(int);
                void ins_bet(int, int);

                int rem_beg();
                int rem_end();
                int rem_bet(int);

                void display();
                void count();
                void reverse();
                void search(int);
}*list;

node* node :: create()
{
```

```
        node *z= new node;
        return(z);
}

void node :: ins_beg(int x)
{
        node *p=list, *q;
        if(p==NULL)
        {
                p=create();
                p->prev=p;
                p->info=x;
                p->next=p;
                list=p;
        }
        else
        {
                while(p->next!=list)
                {
                        p=p->next;
                }
                q=create();
                q->prev=p;
                q->info=x;
                q->next=p->next;
                p->next->prev=q;
                p->next=q;
                list=q;
        }
        cout<<"Insert successfully...\n";
}

void node :: ins_end(int x)
{
        node *p=list, *q;
        if(p==NULL)
        {
                p=create();
                p->prev=p;
```

```
            p->info=x;
            p->next=p;
            list=p;
        }
        else
        {
            while(p->next!=list)
            {
                p=p->next;
            }
            q=create();
            q->prev=p;
            q->info=x;
            q->next=p->next;
            p->next->prev=q;
            p->next=q;
        }
        cout<<"Insert successfully...\n";
}

void node :: ins_bet(int aft, int x)
{
        node *p=list, *q;
        if((p==NULL) || (p->prev==p && p->next==p))
        {
            cout<<"Insert between NOT possible...\n";
        }
        else
        {
            while(p->next!=list)
            {
                if(p->info==aft)
                {
                    q=create();
                    q->prev=p;
                    q->info=x;
                    q->next=p->next;
                    p->next->prev=q;
                    p->next=q;
```

```
                }
                p=p->next;
            }
        }
        cout<<"Insert successfully...\n";
}

int node :: rem_beg()
{
        int z;
        node *p=list, *temp;
        if(p==NULL)
        {
            cout<<"List is empty...\n";
            return(0);
        }
        else if(p->next==p)
        {
            z=p->info;
            delete(p);
            list=NULL;
            return(z);
        }
        else
        {
            while(p->next!=list)
            {
                p=p->next;
            }
            temp=p->next;
            z=temp->info;
            p->next=temp->next;
            temp->next->prev=p;
            list=temp->next;
            delete(temp);
            return(z);
        }
        cout<<"Remove successfully...\n";
}
```

```
int node :: rem_end()
{
	int z;
	node *p=list, *temp;
	if(p==NULL)
	{
		cout<<"List is empty...\n";
		return(0);
	}
	else if(p->next==p)
	{
		z=p->info;
		delete(p);
		list=NULL;
		return(z);
	}
	else
	{
		while(p->next->next!=p)
		{
			p=p->next;
		}
		temp=p->next;
		z=temp->info;
		p->next=temp->next;
		temp->next->prev=p;
		delete(temp);
		return(z);
	}
	cout<<"Remove successfully...\n";
}

int node :: rem_bet(int aft)
{
	int z;
	node *p=list, *temp;
	if(p==NULL)
	{
```

```
            cout<<"List is empty...\n";
            return(0);
        }
        else if(p->next==p || p->next->next==p)
        {
            cout<<"Remove between NOT possible...\n";
            return(0);
        }
        else
        {
            while(p->next!=list)
            {
                if(p->info==aft)
                {
                    temp=p->next;
                    z=temp->info;
                    p->next=temp->next;
                    temp->next->prev=p;
                    delete(temp);
                    return(z);
                }
                p=p->next;
            }
            return(0);
        }
        cout<<"Remove successfully...\n";
}

void node :: display()
{
        node *p=list;
        cout<<"Nodes in list : \n";
        do
        {
            cout<<"\t"<<p->info;
            p=p->next;
        }while(p!=list);
}
```

```
void node :: count()
{
	int c=0;
	node *p=list;
	do
	{
		c++;
		p=p->next;
	}while(p!=list);
	cout<<"Total nodes in list is : \n"<<c;
}

void node :: reverse()
{
	node *t1=list, *t2, *t3=list;
	do
	{
		t2=t1->next;
		t1->next=t3;
		t1->prev=t2;
		t3=t1;
		t1=t2;
	}while(t1!=list);
	list=t3;
	t1->next=t3;
	cout<<"List is reversed...\n";
}

void node :: search(int x)
{
	int t=0;
	node *p=list;
	do
	{
		if(p->info==x)
		{
			t=1;
			break;
		}
```

```
            p=p->next;

        }while(p!=list);

        if(t==1)
        {
            cout<<"Node is found...\n";
        }
        else
        {
            cout<<"Node is not found...\n";
        }
}

void main()
{
        int af, n, ch, z;
        clrscr();
        node obj;
        do
        {
            cout<<"\nEnter your choise : ";
            cout<<"\n \t1.Insert Begin.\n \t2.Insert End\n \t3.Insert
Between\n \t4.Remove Begin\n \t5.Remove End\n \t6.Remove
Between\n \t7.Display\n \t8.Count\n \t9.Reverse\n \t10.Search\n
\t11.Exit\n \t";
            cin>>ch;

            switch(ch)
            {
                case 1:
                    cout<<"Enter value : ";
                    cin>>n;
                    obj.ins_beg(n);
                    break;

                case 2:
                    cout<<"Enter value : ";
                    cin>>n;
```

```
            obj.ins_end(n);
            break;

        case 3:
            cout<<"After which value : ";
            cin>>af;
            cout<<"Enter value : ";
            cin>>n;
            obj.ins_bet(af,n);
            break;

        case 4:
            z=obj.rem_beg();
            cout<<"Removed value : "<<z<<"\n";
            break;

        case 5:
            z=obj.rem_end();
            cout<<"Removed value : "<<z<<"\n";
            break;

        case 6:
            cout<<"After which value : ";
            cin>>af;
            z=obj.rem_bet(af);
            cout<<"Removed value : "<<z<<"\n";
            break;

        case 7:
            obj.display();
            break;

        case 8:
            obj.count();
            break;

        case 9:
            obj.reverse();
            break;
```

```
                case 10:
                        cout<<"Search node : ";
                        cin>>n;
                        obj.search(n);
                        break;

                case 11:
                        cout<<"Exit program...";
                        break;

                default:
                        cout<<"Wrong input...";
                        break;
            }
        }while(ch!=11);

        getch();
}
```
