TABLE OF CONTENTS

FACTORIAL OF GIVEN NUMBER USING RECURSION

Expt.No-1 a) **Date: 15-07-2022**

AIM:

To Find out the factorial of given number using recursion.

PROCEDURE:

Step-1	**Start** the Program
Step-2	**Declare** class factorial
Step-3	**Declare** n as integer
Step-4	**Read** n value
Step-5	**Call** output value
Step-6	**Call** fact() and n as argument value
Step-7	**If** f<=1 exit else goto Step-6
Step-8	**Display** result.
Step-9	**END of the Program**

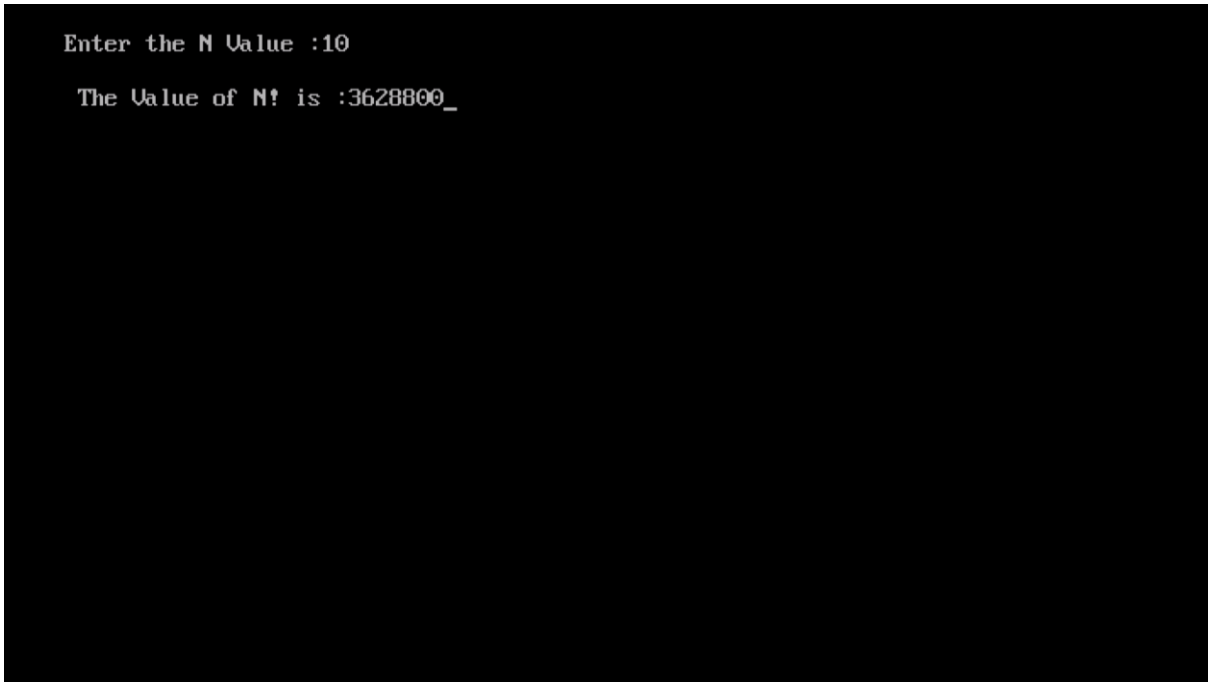
PROGRAM

```
#include<iostream.h>
#include<conio.h>
class factorial
{
        int n;
public:
        void input()
        {
                cout<<"\nEnter the N Value :";
                cin>>n;
        }
```

```
void output()
        {
                double result;
                result= fact(n);
           cout<<"\n The Value of N! is :"<<result;
        }
double fact(double f);
};

double factorial::fact(double f)
{
        if(f<=1)
        {
                return(1);
        }
        else
        {
                return(f*fact(f-1));
        }
}
void main()
{
clrscr();
        factorial f;
        f.input();
        f.output();
getch();
}
```

SAMPLE I/O



RESULT:

The output has been verified successfully.

FIBONNACCI SERIES FOR GIVEN N VALUE USING RECURSION

Expt.No-1 b) **Date: 15-07-2022**

AIM:

To generate the Fibonacci series for given n value using recursion.

PROCEDURE:

Step-1	**Start** the Program
Step-2	**Declare** class fibonnaci
Step-3	**Declare** n as integer
Step-4	**Read** n value
Step-5	**Call** output value
Step-6	**Call** fibo() and n,var1,var2 as argument value
Step-7	**If** nvalue==0 exit else goto Step-6
Step-8	**Display** result.
Step-9	**END of the Program**

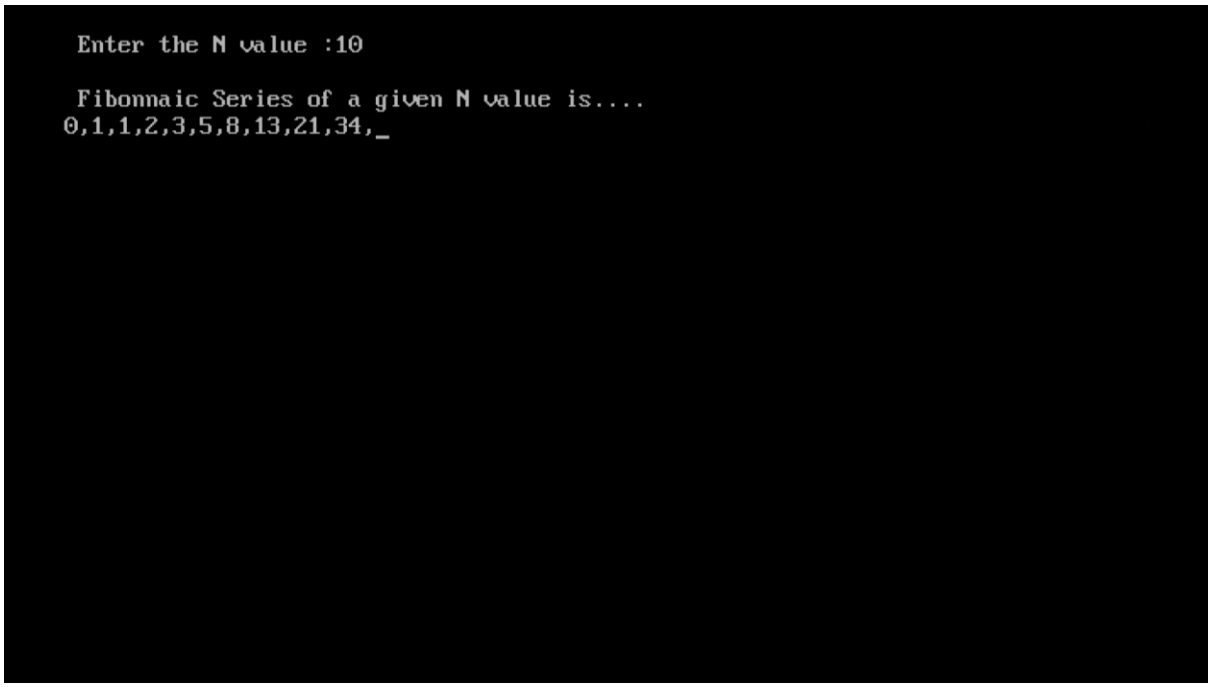
PROGRAM

```
 #include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class fibonnaci
{
        int nvalue;
public:
        void input()
        {
                cout<<"\n Enter the N value :";
                cin>>nvalue;
        }
```

```
        void fibo(long int,long int,long int);
        void output()
        {
                long int var1=-1, var2=1;
                cout<<"\n Fibonnaic Series of a given N value is....\n";
                fibo(nvalue,var1,var2);
        }

};
void main()
{
    fibonnaci f;
clrscr();
    f.input();
    f.output();
}
void fibonnaci::fibo(long int nvalue, long int var1, long int var2)
{
        if(nvalue==0)
        {     getch();
                exit(0);
        }
        cout<<var1+var2<<",";
        fibo(--nvalue, var2, var1+var2);
}
```

SAMPLE I/O



RESULT:

The output has been verified successfully.

STACK USING ARRAYS

Expt.No-2 **Date: 28-07-2022**

AIM:

To implement the concept of Stack using Arrays.

PROCEDURE:

Step-1	**Start** the Program
Step-2	**Declare** Stack_class as class
Step-3	**Declare** Top,stack,n_value as integer
Step-4	**Assign** n value,top in Stack_class as constructor
Step-5	**Read** a choice from User
Step-6	**If** choice==1 Call Push()
Step-7	**Check** whether the size of the stack is full, if full then display **'Stack Overflow'** else add an element to Stack
Step-8	**If** choice==2 Call Pop()
Step-9	**Check** whether the stack contains elements or not, if empty then display **'Stack Underflow'** else remove the last element from Stack
Step-10	**If** choice==3 Call View()
Step-11	**Display** the contents of Stack.
Step-12	**END of the Program**

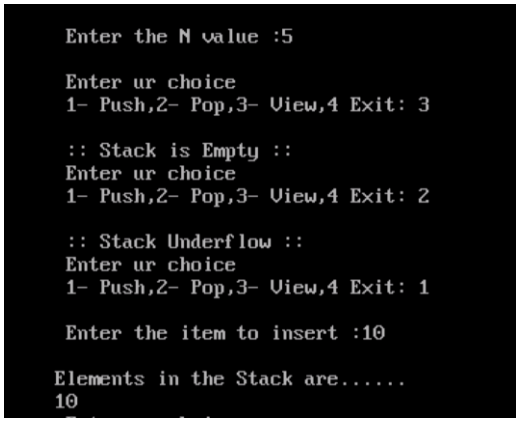
PROGRAM

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class Stack_class
{
        int stack[100],top,nvalue;
public:
        Stack_class()
        {
                top=-1;
                cout<<"\n Enter the N value :";
                cin>>nvalue;
        }
        void Push();
```

```
        void Pop();
        void View();
};
void Stack_class::Push()
{
        if(top>=nvalue-1)
        {
                cout<<"\n :: Stack Overflow ::";
        }
        else
        {
                top++;
                cout<<"\n Enter the item to insert :";
                cin>>stack[top];
        }
        View();
}
void Stack_class::Pop()
{
        if(top<0)
        {
                cout<<"\n :: Stack Underflow ::";
        }
        else
        {
                cout<<"\n Deleted Item= "<<stack[top];
                top--;
                View();
        }
}
void Stack_class::View()
{
        int i;
        if(top==-1)
        {
                cout<<"\n :: Stack is Empty ::";
        }
        else
        {
                cout<<"\nElements in the Stack are.....\n";
                for(i=top;i>=0;i--)
                {
                        cout<<stack[i]<<"\t";
                }
        }
}
```

```
void main()
{
        int choice;
clrscr();
        Stack_class S;
        while(1)
        {
                cout<<"\n Enter ur choice\n 1- Push,2- Pop,3- View,4 Exit: ";
                cin>>choice;
                if(choice==1)
                {
                        S.Push();
                }
                else if(choice==2)
                {
                        S.Pop();
                }
                else if(choice==3)
                {
                        S.View();
                }
                else
                {
                        getch();
                        exit(0);;
                }
        }
}
```

SAMPLE I/O



```
 Enter ur choice
 1- Push,2- Pop,3- View,4 Exit: 1

 Enter the item to insert :20

Elements in the Stack are......
20      10
 Enter ur choice
 1- Push,2- Pop,3- View,4 Exit: 1

 Enter the item to insert :30

Elements in the Stack are......
30      20      10
 Enter ur choice
 1- Push,2- Pop,3- View,4 Exit: 1

 Enter the item to insert :40

Elements in the Stack are......
40      30      20      10
 Enter ur choice
 1- Push,2- Pop,3- View,4 Exit:         1

 Enter the item to insert :50

Elements in the Stack are......
50      40      30      20      10
 Enter ur choice
 1- Push,2- Pop,3- View,4 Exit: 1

 :: Stack Overflow ::
Elements in the Stack are......
50      40      30      20      10
 Enter ur choice
 1- Push,2- Pop,3- View,4 Exit: 3

Elements in the Stack are......
50      40      30      20      10
 Enter ur choice
 1- Push,2- Pop,3- View,4 Exit: 2

 Deleted Item= 50
Elements in the Stack are......
40      30      20      10
 Enter ur choice
 1- Push,2- Pop,3- View,4 Exit: 2

 Deleted Item= 40
Elements in the Stack are......
30      20      10
 Enter ur choice
 1- Push,2- Pop,3- View,4 Exit: 2

 Deleted Item= 30
Elements in the Stack are......
20      10
 Enter ur choice
 1- Push,2- Pop,3- View,4 Exit: 2

 Deleted Item= 20
Elements in the Stack are......
10
 Enter ur choice
 1- Push,2- Pop,3- View,4 Exit: 2

 Deleted Item= 10
 :: Stack is Empty ::
```

```
 Enter ur choice
 1- Push,2- Pop,3- View,4 Exit: 1

 Enter the item to insert :15

Elements in the Stack are......
15
 Enter ur choice
 1- Push,2- Pop,3- View,4 Exit: 4
```

RESULT:

The output has been verified successfully.

QUEUE USING ARRAYS

Expt.No-3 **Date: 04-08-2022**

AIM:

To implement the concept of Queue using Arrays.

PROCEDURE:

Step-1	**Start** the Program
Step-2	**Declare** Queue as class
Step-3	**Declare** front,value[100],n as integer
Step-4	**Assign** n,front,rear values in constructor
Step-5	**Read** a choice from User
Step-6	**If** choice==1 Call Insert()
Step-7	**Check** whether the size of the Queue is full, if full then display **'Queue Overflow'** else add an element to Queue
Step-8	**If** choice==2 Call Delete()
Step-9	**Check** whether the Queue contains elements or not, if empty then display **'Queue Underflow'** else remove the first element from Queue
Step-10	**If** choice==3 Call View()
Step-11	**Display** the contents of Queue.
Step-12	**END of the Program**

PROGRAM

```
 #include<iostream.h>
#include<stdlib.h>
#include<conio.h>
class Queue
{
        int value[100],rear,front,n;
public:
        Queue()
        {
                rear=front=0;
                cout<<"\n Enter the N value :";
                cin>>n;
        }
        void Insert();
```

```
        void Delete();
        void View();
};
void Queue::Insert()
{
        if(rear==n)
        {
                cout<<"\n\n :: Queue Overflow ::";

        }
        else
        {
                rear++;
                cout<<"\n \nEnter the item to insert :";
                cin>>value[rear];
        }
        if(front==0)
        {
                front=1;
        }
        View();
}

void Queue::Delete()
{
        if(front>rear || front==0)
        {
                cout<<"\n :: Queue Underflow ::";
        }
        else
        {
                cout<<"\n \n Deleted Item= "<<value[front];
                front++;
                View();
        }
        if(rear<front)
        {
                front=rear=0;
        }
}
void Queue::View()
{
        int i;

        if(front==0 || rear<front)
        {
```

```
                cout<<"\n :: Queue is Empty ::";
        }
        else
        {
                cout<<"\nThe Contents in Queue are....."<<endl;
                for(i=rear;i>=front;i--)
                {
                        cout<<value[i]<<"\t";
                }
        }
}
void main()
{
clrscr();
        int choice;
        Queue Q;

        while(1)
        {
                cout<<"\n Enter ur choice";
                cout<<"\n 1- Insert,2- Delete,3- View,4 Exit: ";
                cin>>choice;
                if(choice==1)
                {
                        Q.Insert();
                }
                else if(choice==2)
                {
                        Q.Delete();
                }
                else if(choice==3)
                {
                        Q.View();
                }
                else
                {
                        getch();
                        exit(0);;
                }
        }
}
```

SAMPLE I/O

```
Enter the N value :5
 Enter ur choice
 1- Insert,2- Delete,3- View,4 Exit: 2
 :: Queue Underflow ::

 Enter ur choice
 1- Insert,2- Delete,3- View,4 Exit: 3
 :: Queue is Empty ::

 Enter ur choice
 1- Insert,2- Delete,3- View,4 Exit: 1

Enter the item to insert :10
The Contents in Queue are.....10

 Enter ur choice
 1- Insert,2- Delete,3- View,4 Exit: 1

Enter the item to insert : 20
The Contents in Queue are.....20        10

 Enter ur choice
 1- Insert,2- Delete,3- View,4 Exit: 1

Enter the item to insert : 30
The Contents in Queue are.....30        20        10

 Enter ur choice
 1- Insert,2- Delete,3- View,4 Exit: 1

Enter the item to insert : 40
The Contents in Queue are.....40        30        20        10

 Enter ur choice
 1- Insert,2- Delete,3- View,4 Exit: 1

Enter the item to insert :50
The Contents in Queue are.....50        40        30        20        10

 Enter ur choice
 1- Insert,2- Delete,3- View,4 Exit: 1

 :: Queue Overflow ::
The Contents in Queue are.....50        40        30        20        10

 Enter ur choice
 1- Insert,2- Delete,3- View,4 Exit: 3
The Contents in Queue are.....50        40        30        20        10
```

```
Enter ur choice
1- Insert,2- Delete,3- View,4 Exit: 2

Deleted Item=10
The Contents in Queue are.....50        40        30        20

Enter ur choice
1- Insert,2- Delete,3- View,4 Exit: 2

Deleted Item=20
The Contents in Queue are.....50        40        30

Enter ur choice
1- Insert,2- Delete,3- View,4 Exit: 2

Deleted Item=30
The Contents in Queue are.....50        40

Enter ur choice
1- Insert,2- Delete,3- View,4 Exit: 2

Deleted Item=40
The Contents in Queue are.....50

Enter ur choice
1- Insert,2- Delete,3- View,4 Exit: 2

Deleted Item=50
:: Queue is Empty ::

Enter ur choice
1- Insert,2- Delete,3- View,4 Exit: 2
:: Queue Underflow ::

Enter ur choice
1- Insert,2- Delete,3- View,4 Exit: 3
:: Queue is Empty ::

Enter ur choice
1- Insert,2- Delete,3- View,4 Exit: 4
```

RESULT:

The output has been verified successfully.

SINGLE LINKED LIST

Expt.No-4 **Date: 17-08-2022**

AIM:

To create a single linked list and perform insert, delete and traversal operations on that list.

PROCEDURE:

Step-1	**Start** the Program
Step-2	**Declare List** as class
Step-3	**Declare** a structure node with two members data and link
Step-4	**Declare** Start node and **Assign** as NULL
Step-5	**Read** a choice from User
Step-6	**If** choice==1 Call Insert ()
Step-7	**Insert** an element in the list depending on user choice
Step-8	**If** choice==2 Call Delete ()
Step-9	**Delete** an element in the list depending on user choice
Step-10	**If** choice==3 Call View ()
Step-11	**Display** the contents of Single Linked List.
Step-12	**END of the Program**

PROGRAM

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class List
{
        struct node
        {
                int data;
                node *link;
        }*start;
public:
        void Insert();
        void Delete();
        void View();
        List()
```

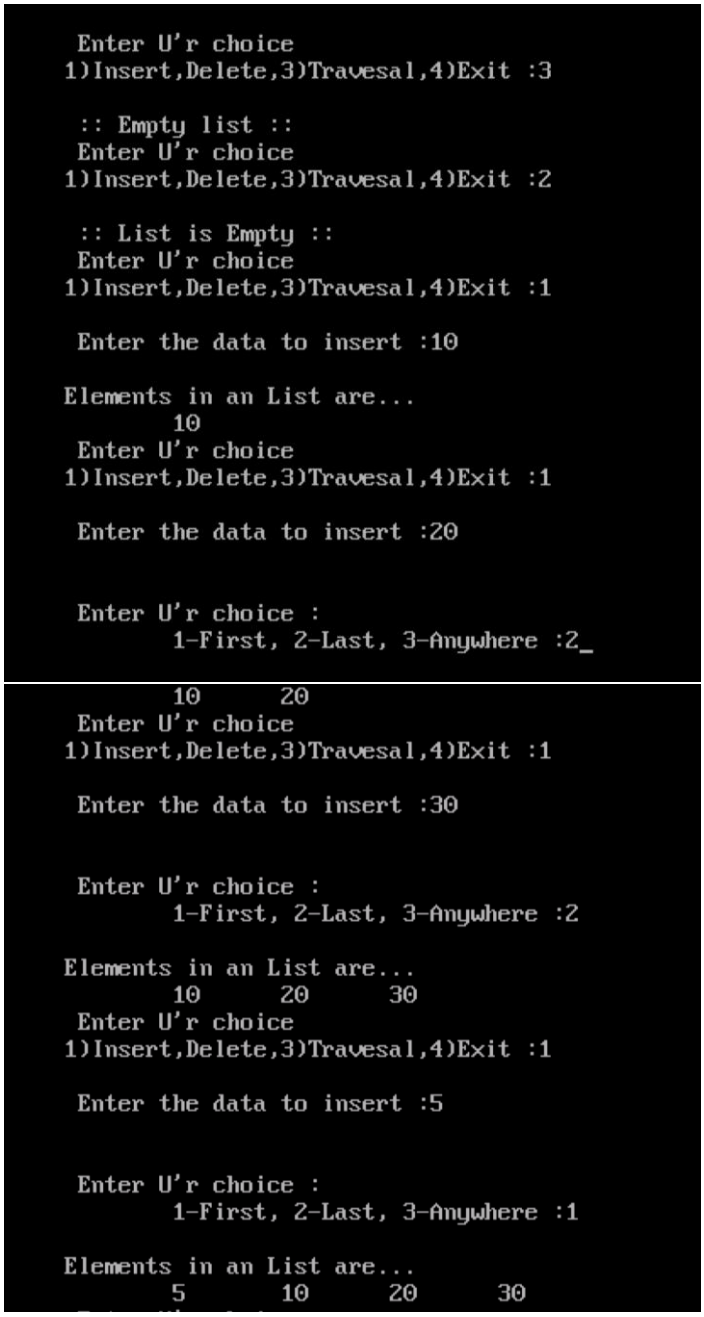
```
	{
		start=NULL;
	}
};
void List::Insert()
{
	int ch,item;
	node *temp,*nnode;
	nnode=new node;
	cout<<"\n Enter the data to insert :";
	cin>>nnode->data;
	nnode->link=NULL;
	if(start==NULL)
	{
		start=nnode;
	}
	else
	{
		cout<<"\n\n Enter U'r choice :";
		cout<<"\n\t1-First, 2-Last, 3-Anywhere :";
		cin>>ch;
		if(ch==1)
		{
			nnode->link=start;
			start=nnode;
		}
		else if(ch==2)
		{
			temp=start;
			while(temp->link!=NULL)
			{
				temp=temp->link;
			}
			temp->link=nnode;
		}
		else
		{
			cout<<"\n Enter the data after to insert :";
			cin>>item;
			temp=start;
			while(temp!=NULL)
			{
				if(temp->data==item)
				{
					break;
				}
```

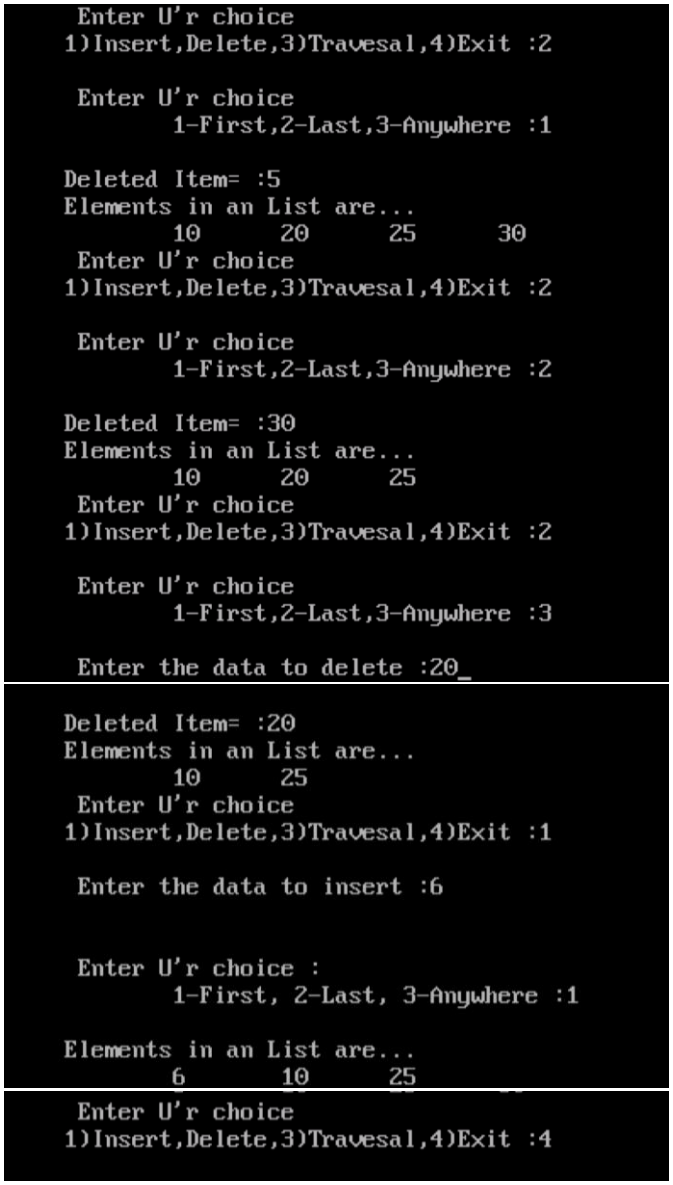
```
                        temp=temp->link;
                    }
                    if(temp==NULL)
                    {
                        cout<<"\n :: Node does't exists ::";
                    }
                    else
                    {
                        nnode->link=temp->link;
                        temp->link=nnode;
                    }
                }
            }
}
void List::Delete()
{
        int ch,item;
        node *temp, *prev=NULL;
        if(start==NULL)
        {
            cout<<"\n :: List is Empty ::";
        }
        else
        {
            cout<<"\n Enter U'r choice";
            cout<<"\n\t1-First,2-Last,3-Anywhere :";
            cin>>ch;
            if(ch==1)
            {   cout<<"\nDeleted Item= :"<<start->data;
                start=start->link;
            }
            else if(ch==2)
            {
                temp=start;
                while(temp->link!=NULL)
                {
                    prev=temp;
                    temp=temp->link;
                }
                if(prev==NULL)
                {
                    cout<<"\nDeleted Item= :"<<start->data;
                    start=NULL;
                }
                else
                {
```

```
                        cout<<"\nDeleted Item= :"<<temp->data;
                        prev->link=NULL;
                    }
                }
                else if(ch==3)
                {
                    cout<<"\n Enter the data to delete :";
                    cin>>item;
                    if(start->link==NULL)
                    {
                        cout<<"\nDeleted Item= :"<<start->data;
                        start=NULL;
                    }
                    else
                    {
                        temp=start;
                        while(temp!=NULL)
                        {
                            if(temp->data==item)
                            {
                                break;
                            }
                            prev=temp;
                            temp=temp->link;
                        }
                        if(prev==NULL)
                        {
                            cout<<"\nDeleted Item= :"<<start->data;
                            start=start->link;
                        }
                        else if(temp!=NULL)
                        {
                            cout<<"\nDeleted Item= :"<<temp->data;
                            prev->link=temp->link;
                            delete temp;
                        }
                        else
                        {
                            cout<<"\n :: Node does n't exists ::";
                        }
                    }
                }
                View();
            }
}
void List::View()
```

```
{
        node *temp;
        if(start==NULL)
        {
                cout<<"\n :: Empty list ::";
        }
        else
        {
                cout<<"\nElements in an List are...\n";
                temp=start;
                while(temp!=NULL)
                {
                        cout<<"\t"<<temp->data;
                        temp=temp->link;
                }
        }
}
void main()
{
        int ch;
        List l;
clrscr();
        do
        {
                cout<<"\n Enter U'r choice\n1)Insert,Delete,3)Travesal,4)Exit :";
                cin>>ch;
                if(ch==1)
                {
                        l.Insert();
                        l.View();
                }
                else if(ch==2)
                {
                        l.Delete();
                }
                else if(ch==3)
                {
                        l.View();
                }
                else if(ch==4)
                {
                        break;
                }
        }while(ch<5);
getch();
}
```

SAMPLE I/O


Enter U'r choice
1)Insert,Delete,3)Travesal,4)Exit :3
:: Empty list ::
Enter U'r choice
1)Insert,Delete,3)Travesal,4)Exit :2
:: List is Empty ::
Enter U'r choice
1)Insert,Delete,3)Travesal,4)Exit :1
Enter the data to insert :10
Elements in an List are...
10
Enter U'r choice
1)Insert,Delete,3)Travesal,4)Exit :1
Enter the data to insert :20
Enter U'r choice :
1-First, 2-Last, 3-Anywhere :2
10 20
Enter U'r choice
1)Insert,Delete,3)Travesal,4)Exit :1
Enter the data to insert :30
Enter U'r choice :
1-First, 2-Last, 3-Anywhere :2
Elements in an List are...
10 20 30
Enter U'r choice
1)Insert,Delete,3)Travesal,4)Exit :1
Enter the data to insert :5
Enter U'r choice :
1-First, 2-Last, 3-Anywhere :1
Elements in an List are...
5 10 20 30



RESULT:

The Linked list has been implemented successfully.

BINARY TREE TRAVERSALS

Expt.No-5 **Date: 31-08-2022**

AIM:

To perform binary tree operations in a tree.

PROCEDURE:

Step-1	**Start** the Program
Step-2	**Create** a tree with 3 fields, data, left and right. Data contains actual data and left contains address of the left sub tree and right contains the address of right subtree.
Step-3	**Initialize** the data members using Constructor Function.
Step-4	**Insert** values into tree based on user choice.
Step-5	After Performing Insert operation, Traversal into Tree. Traversal can be done in three ways. **Inorder**: Left, Root, Right **PreOrder**: Root, Left, Right **PostOrder**: Left, Right, Root.
Step-6	**END of the Program**

PROGRAM

```
 #include<iostream.h>
#include<conio.h>
class BinaryTree
{
        struct node
        {
                int data;
                node *left,*right;
        }*root;
public:
        void Inorder(node *);
        void Preorder(node *);
        void Postorder(node *);
        void Input();
        void Display();
        BinaryTree()
        {
                root=NULL;
        }
```

```
};
void BinaryTree::Input()
{
	int flag=0;
	node *temp,*nnode,*prev;
	nnode = new node;
	cout<<"\n Enter the Data to Insert :";
	cin>>nnode->data;
	nnode->left=nnode->right=NULL;
	if(root==NULL)
	{
		root=nnode;
	}
	else
	{
		char choice;
		temp= root;
		while(temp!=NULL)
		{
			cout<<"\n Enter ur Choice whether to insert in 0-right, 1- left:";
			cin>>choice;
			prev=temp;
			if(choice=='l' || choice=='L')
			{
				temp= temp->left;
			}
			else if(choice=='r' ||choice=='R')
			{
				temp= temp->right;
			}
			else
			{
				cout<<"\n Invalid Selection";
				flag=1;
			}
		}
		if(flag==0)
		{
			if(choice=='l'|| choice=='L')
			{
				prev->left=nnode;
			}
			else if(choice=='r'||choice=='R')
			{
				prev->right=nnode;
			}
```

```
                }
        }
}
void BinaryTree::Display()
{
        cout<<"\n Inorder Travesal :";
        Inorder(root);
        cout<<"\n Preorder Travesal :";
        Preorder(root);
        cout<<"\n Postorder Travesal :";
        Postorder(root);
}

void BinaryTree::Inorder(struct node *temp)
{
        if(temp==NULL)
        {
                cout<<"\n Empty Tree";
        }
        else
        {
                if(temp->left!=NULL)
                {
                        Inorder(temp->left);
                }
                cout<<temp->data<<"-->";
                if(temp->right!=NULL)
                {
                        Inorder(temp->right);
                }
        }
}

void BinaryTree::Preorder(node *temp)
{
        if(temp==NULL)
        {
                cout<<"\n Empty Tree";
        }
        else
        {
                cout<<temp->data<<"-->";
                if(temp->left!=NULL)
                {
                        Preorder(temp->left);
                }
```

```
                if(temp->right!=NULL)
                {
                        Preorder(temp->right);
                }
        }
}

void BinaryTree::Postorder(node *temp)
{
        if(temp==NULL)
        {
                cout<<"\n Empty Tree";
        }
        else
        {
                if(temp->left!=NULL)
                {
                        Postorder(temp->left);
                }
                if(temp->right!=NULL)
                {
                        Postorder(temp->right);
                }
                cout<<temp->data<<"-->";
        }
}

void main()
{
        int selection;
        BinaryTree bt;
clrscr();
        cout<<"\n Implementation of Binary Trees\n\n\n\n";

        do
        {
                cout<<"\n Enter ur Choice \n 1- Insert \n 2- Display \n 3- Exit :";
                cin>>selection;
                if(selection==1)
                {
                        bt.Input();
                }
                else if(selection==2)
                {
                        bt.Display();
                }
```

```
                else
                {
                        break;
                }
        }while(selection<4);
getch();
}
```

SAMPLE I/O

```
                Implementation of Binary Trees

Enter ur Choice
1- Insert
2- Display
3- Exit :1

Enter the Data to Insert :10

Enter ur Choice
1- Insert
2- Display
3- Exit :1

Enter the Data to Insert :3

Enter ur Choice whether to insert in R-right, L- left:l
```

```
Enter ur Choice
1- Insert
2- Display
3- Exit :1

Enter the Data to Insert :30

Enter ur Choice whether to insert in R-right, L- left:l

Enter ur Choice whether to insert in R-right, L- left:l

Enter ur Choice
1- Insert
2- Display
3- Exit :1
```

```
Enter the Data to Insert :21

Enter ur Choice whether to insert in R-right, L- left:l

Enter ur Choice whether to insert in R-right, L- left:r

Enter ur Choice
1- Insert
2- Display
3- Exit :1
```

```
Enter ur Choice
1- Insert
2- Display
3- Exit :1

Enter the Data to Insert :20

Enter ur Choice whether to insert in R-right, L- left:r

Enter ur Choice
1- Insert
2- Display
3- Exit :1

Enter the Data to Insert :8

Enter ur Choice whether to insert in R-right, L- left:r

Enter ur Choice whether to insert in R-right, L- left:l
```

```
Enter ur Choice
1- Insert
2- Display
3- Exit :1

Enter the Data to Insert :    31

Enter ur Choice whether to insert in R-right, L- left:r

Enter ur Choice whether to insert in R-right, L- left:r

Enter ur Choice
1- Insert
2- Display
3- Exit :2

Inorder Travesal :30-->3-->21-->10-->8-->20-->31-->
Preorder Travesal :10-->3-->30-->21-->20-->8-->31-->
Postorder Travesal :30-->21-->3-->8-->31-->20-->10-->
Enter ur Choice
1- Insert
2- Display
3- Exit :3_
```

RESULT:

The Tree Traversals has been implemented successfully.

QUICK SORT

Expt.No-6 **Date: 07-09-2022**

AIM:

To perform sorting operations using Quick sort algorithm for the given elements .

PROCEDURE:

Step-1	**Start** the Program
Step-2	**Assign** elements in to array using constructor.
Step-3	**Assign** Pivot element and **divide** the elements around pivot element
Step-4	**Sort** the elements based on the pivot element
Step-5	**Display** the sorted Array.
Step-6	**END of the Program**

PROGRAM

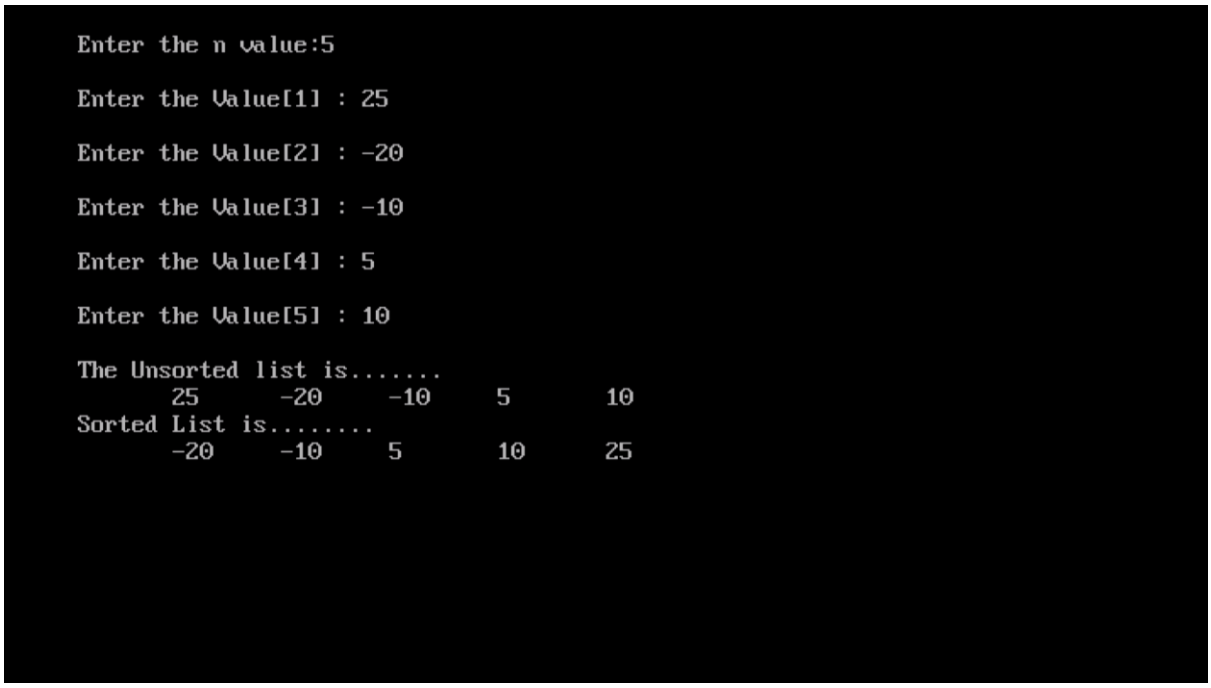
```
 #include<iostream.h>
#include<conio.h>
class QSort
{
        int value[50],n,i;
public:
        void Quick(int value[],int,int);
        void Swap(int value[],int,int);
        void Display();
        void Sort();
        QSort();
};
QSort::QSort()
{
        cout<<"\n Enter the n value:";
        cin>>n;
        for(i=0;i<n;i++)
        {
                cout<<"\n Enter the Value["<<i+1<<"] : ";
                cin>>value[i];
        }
}
void QSort::Sort()
{
        Quick(value,0,n-1);
}
```

```
void QSort::Quick(int value[],int first,int last)
{
	int i,j,pivot,temp;
	if(first<last)
	{
		pivot=value[first];
		i=first;
		j=last;
		while(i<j)
		{
			while(value[i]<=pivot && i <last)
			{
				i++;
			}
			while(value[j]>=pivot && j>first)
			{
				j--;
			}
			if(i<j)
			{
				Swap(value,i,j);
			}
		}
		Swap(value,first,j);
		Quick(value,first,j-1);
		Quick(value,j+1,last);
	}
}
void QSort::Swap(int value[],int i,int j)
{
	int temp;
	temp=value[i];
	value[i]=value[j];
	value[j]=temp;
}
void QSort::Display()
{
	for(i=0;i<n;i++)
	{
		cout<<"\t"<<value[i];
	}

}
void main()
{
clrscr();
```

```
        QSort Q;
        int i;
        cout<<"\n The Unsorted list is.....\n";
        Q.Display();
        cout<<"\n Sorted List is.....\n";
        Q.Sort();
        Q.Display();
getch();
}
```

SAMPLE I/O



RESULT:

The Quick Sort has been implemented successfully.

MERGE SORT

Expt.No-7 **Date: 14-09-2022**

AIM:

To perform sorting operations using Merge sort algorithm.

PROCEDURE:

Step-1	**Start** the Program
Step-2	**Assign** elements in to array using constructor.
Step-3	**Split** the array into Two equal parts
Step-4	**Sort** the elements based on the starting and ending index and merge the array elements
Step-5	**Display** the sorted Array.
Step-6	**END of the Program**

PROGRAM

```
 #include<iostream.h>
#include<conio.h>
class MSort
{
        int valueA[50],temp[50],size,i;
public:
        void Merge_split(int valueA[],int first,int last);
        void Merge(int valueA[],int first_a,int last_a,int first_b,int last_b);
        void Display();
        void Sort();
        MSort();
};
MSort::MSort()
{
        cout<<"\n Enter the Size :";
        cin>>size;

        for(i=0;i<size;i++)
        {
                cout<<"\n Enter the Element: ";
                cin>>valueA[i];
        }
}
void MSort::Display()
{
```

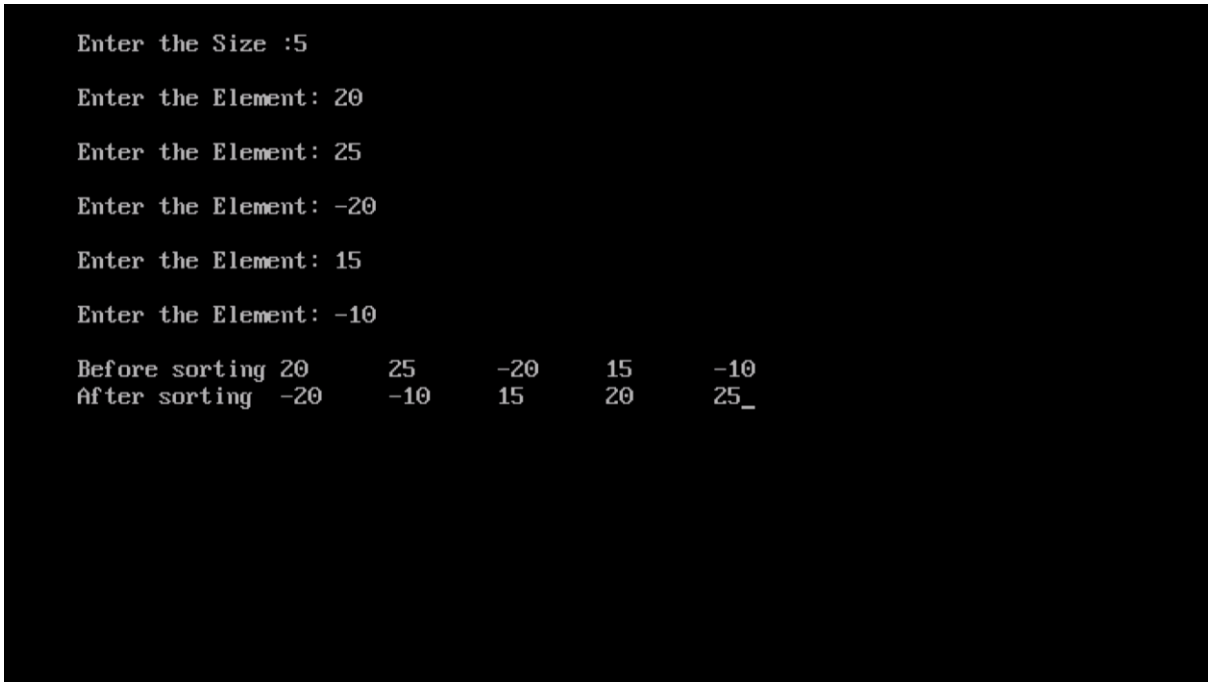
```
	for(i=0;i<size;i++)
	{
		cout<<"\t"<<valueA[i];
	}

}
void MSort::Sort()
{
	Merge_split(valueA,0,size-1);
}
void main()
{
	int f_lvar,size;
clrscr();
	MSort M;
	cout<<"\n Before sorting";
	M.Display();
	M.Sort();
	cout<<"\n After sorting";
	M.Display();
getch();
}
void MSort::Merge_split(int valueA[],int first,int last)
{
	int mid;
	if(first<last)
	{
		mid=(first + last)/2;
		Merge_split(valueA,first,mid);
		Merge_split(valueA,mid+1,last);
		Merge(valueA,first,mid,mid+1,last);
	}
}
void MSort::Merge(int valueA[],int first_a,int last_a,int first_b,int last_b)
{
	int i,j,position=0;

	i=first_a;
	j=first_b;
	while(i<=last_a && j<=last_b)
	{
		if(valueA[i]<valueA[j])
		{
			temp[position]=valueA[i++];
		}
		else
```

```
                {
                        temp[position]=valueA[j++];
                }
                position++;
        }
        while(i<=last_a)
        {
                temp[position++]=valueA[i++];
        }
        while(j<=last_b)
        {
                temp[position++]=valueA[j++];
        }
        i=first_a;
        j=0;
        while(i<=last_b && j<= position)
        {
                valueA[i++]=temp[j++];
        }
}
```

SAMPLE I/O



RESULT:

The Merge Sort has been implemented successfully.

KNAPSACK PROBLEM

Expt.No-7 **Date: 21-09-2022**

AIM:

To demonstrate knapsack problem using Greedy algorithm.

PROCEDURE:

Step-1	**Start** the Program
Step-2	**Assign** elements in to profit and weight using constructor.
Step-3	**Calculate** the x[i] value
Step-4	**Fill** the Bag from highest x[i] to lowest.
Step-5	**Calculate** the Sum of X[i] and Weight[i]
Step-6	**END of the Program**

PROGRAM

```
 #include<iostream.h>
#include<iomanip.h>
#include<conio.h>
class Greedy
{
        int i,j,n,temp,index[20];
        float profit[20], weight[20], x[20], max, capacity;
public:
        Greedy();
        void Display();
};
void main()
{
clrscr();
        Greedy G;
        G.Display();
getch();
}
Greedy:: Greedy()
{
        cout<<"\n Enter the Maximum weight of the BAG:";
        cin>>max;
        cout<<" Enter the number of objects :";
        cin>>n;
        for(i=0; i<n; i++)
        {
```

```
		cout<<"\n Enter weight of "<<i+1<<" item w["<<i+1<<"] :";
		cin>>weight[i];
		cout<<"Enter profit of "<<i+1<< " item p["<<i+1<<"] :";
		cin>>profit[i];
	}
	for(i=0; i<n; index[temp]=i, i++)
	{
		for(temp=0, j=0; j<n; j++)
		{
			if((i != j) && (profit[i]/ weight[i]) < (profit[j] / weight[j]))
			{
				temp++;
			}
		}
	}
	capacity = max;
	for(i=0; i<n; i++)
	{
		if(weight[index[i]] > capacity)
		{
			break;
		}
		x[index[i]] = 1.0;
		capacity = capacity - weight[index[i]];
	}
	if(i < n)
	{
		x[index[i]] = capacity / weight[index[i]];
	}
}
void Greedy::Display()
{
	float profit_value=0.0, max_cap =0.0;
	for(i=0;i<n;i++)
	{
		profit_value = profit_value + x[i] * profit[i];
	}
	for(i=0; i<n; i++)
	{
		max_cap = max_cap +(weight[i] * x[i]);
	}
	cout<<"\n The Optimal solution is...\n";
	cout<<"\t\t"<<"Weight"<<"\t"<<"Profit"<<"\t"<<"X\n";
	cout<<"\t"<<"i"<<"\t"<<"w[i]"<<"\t"<<"p[i]"<<"\t"<<"x[i]";
	for(i=0; i<n; i++)
	{
```

```
        cout<<endl<<"\t"<<i+1<<"\t"<<weight[i]<<"\t"<<profit[i]<<"\t"<<x[i];
        }
        cout<<"\n\nThe Sum of p[i] * x[i] = "<<profit_value;
        cout<<"\n The Sum of w[i] * x[i]= ";
        cout<<max_cap;
}
```

SAMPLE I/O

```
Enter the Maximum weight of the BAG:10
Enter the number of objects :6

Enter weight of 1 item w[1] :5
Enter profit of 1 item p[1] :12

Enter weight of 2 item w[2] :4
Enter profit of 2 item p[2] :10

Enter weight of 3 item w[3] :4
Enter profit of 3 item p[3] :2

Enter weight of 4 item w[4] :3
Enter profit of 4 item p[4] :10

Enter weight of 5 item w[5] :2
Enter profit of 5 item p[5] :7

Enter weight of 6 item w[6] :1
Enter profit of 6 item p[6] :6
```

```
The Optimal solution is...
              Weight  Profit  X
       i      w[i]    p[i]    x[i]
       1      5       12      0
       2      4       10      1
       3      4       2       0
       4      3       10      1
       5      2       7       1
       6      1       6       1

The Sum of p[i] * x[i] = 33
 The Sum of w[i] * x[i]= 10_
```

RESULT:

Thus, Knapsack Problem has been implemented successfully.

SHORTEST PATH USING DIJKSTRA ALGORITHM

Expt.No-9 **Date: 26-09-2022**

AIM:

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm

PROCEDURE:

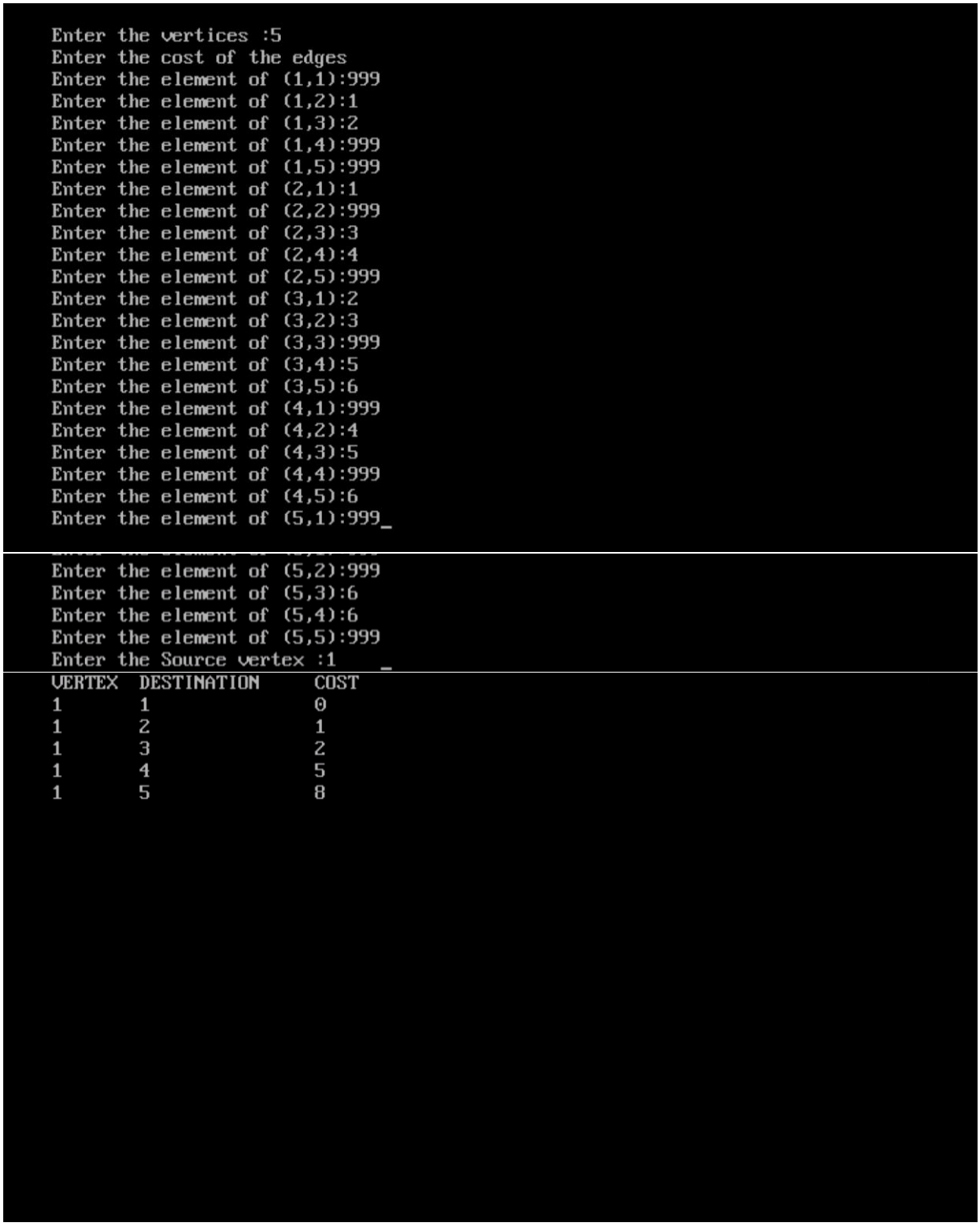
Step-1	**Start** the Program
Step-2	Let G = (A, B) where A = set of vertices
Step-3	**Initially,** let V = {a} and U = V − {a}
Step-4	Let U be the **unvisited** and V be the **visited vertices**
Step-5	Let Dist[t] = w[(a, t)] for every vertex a of A
Step-6	Select the vertex in U that has the smallest value Dist[x]. Let x denote this vertex.
Step-7	If x is the vertex we wish to reach from a, goto 9. If not, let V = V − {x} and U = U − {x}
Step-8	For every vertex t in A, compute Dist[t] with respect to V as, Dist[t] = min{Dist[t], Dist[t] + w (x,t)}
Step-9	Repeat steps 5, 6, and 7
Step-10	**END of the Program**

PROGRAM

```
#include<iostream.h>
#include<conio.h>
class Graph
{
int n, cost[15][15], i, j, s[15], v, u, w, dist[15], num, min;
public:
        Graph();
        void Travesal();
};
Graph::Graph()
{
        cout<<"\nEnter the vertices :";
        cin>>n;
        cout<<"Enter the cost of the edges\n";
        for (i = 1; i <= n; i++)
        {
                for (j = 1; j <= n; j++)
                {       cout<<"Enter the element of ("<<i<<","<<j<<"):";
                        cin>>cost[i][j];
```

```
                }
        }
        cout<<"Enter the Source vertex :";
        cin>>v;
}
void Graph::Travesal()
{
        for (i = 1; i <= n; i++)
        {
                s[i] = 0;
                dist[i] = cost[v][i];
        }
        s[v] = 1;
        dist[v] = 0;
        for (num = 2; num <= n-1; num++)
        {
                min = 999;
                for (w = 1; w <= n; w++)
                        if (s[w] == 0 && dist[w] < min)
                        {
                                min = dist[w];
                                u = w;
                        }
                s[u] = 1;
                for (w = 1; w <= n; w++)
                {
                        if (s[w] == 0)
                        {
                                if (dist[w] > (dist[u] + cost[u][w]))
                                        dist[w] = (dist[u] + cost[u][w]);
                        }
                }
        }
        cout<<"VERTEX\tDESTINATION\tCOST\n";
        for (i = 1; i <= n; i++) {
                cout<<v<<"\t"<<i<<"\t\t"<<dist[i]<<endl;
        }
}
void main ()
{
clrscr();
        Graph G;
clrscr();
        G.Travesal();
getch();
}
```

SAMPLE I/O



RESULT:

Thus, Shortest path has been obtained successfully using Dijkstra Algorithm.

TRAVELLING SALESMAN PROBLEM USING DYNAMIC PROGRAMMING

Expt.No-10 **Date: 29-09-2022**

AIM:

To Implement Travelling Sales Person problem using Dynamic programming.

PROCEDURE:

Step-1	**Start** the Program
Step-2	**Assign** C ({1}, 1) = 0
Step-3	for s = 2 to n do for all subsets S Є {1, 2, 3, … , n} of size s and containing 1 C (S, 1) = ∞
Step-4	for all j Є S and j ≠ 1 C (S, j) = min {C (S – {j}, i) + d(i, j)
Step-5	for i Є S and i ≠ j} Return minj C ({1, 2, 3, …, n}, j) + d(j, i)
Step-6	**END of the Program**

PROGRAM

```
#include<iostream.h>
#include<conio.h>
class Graph
{
        int cost[20][20],path[20],i,j,n;
public:
        Graph();
        int Travesal(int path[],int start,int n);
        void Display();
};
void Graph::Display()
{
        int mincost=Travesal(path,1,n);
        cout<<"\nTravelling Salesman Tour\n";
        for(i=1;i<=n;i++)
        {
                cout<<path[i]<<"--->";
        }
        cout<<"1"<<endl;
        cout<<"Tour Cost="<<mincost;

}
Graph::Graph()
{
```

```
        cout<<"\n Enter the No of Cities:";
        cin>>n;
        cout<<" Enter the Cost Matrix:\nd";
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        cout<<"Enter the Value of ["<<i<<"]["<<j<<"]:";
                        cin>>cost[i][j];
                }
        }
        for(i=1;i<=n;i++)
                path[i]=i;

}
int Graph::Travesal(int path[],int start,int n)
{
        int i,j,k,ccost;
        int mintour[20];
        int temp[20];
        if(start==n-1)
                return cost[path[n-1]][path[n]]+cost[path[n]][1];
        int mincost=999;
        for(i=start+1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        temp[j]=path[j];
                }
                temp[start+1]=path[i];
                temp[i]=path[start+1];
                if(cost[path[start]][path[i]]+(ccost=Travesal(temp,start+1,n))<mincost)
                {
                        mincost=cost[path[start]][path[i]]+ccost;
                        for(k=1;k<=n;k++)
                        mintour[k]=temp[k];
                }
        }
        for(i=1;i<=n;i++)
        {
                path[i]=mintour[i];
        }
return(mincost);
}
void main()
{
```

```
clrscr();
        Graph obj;
        obj.Display();
getch();
}
```

SAMPLE I/O

```
 Enter the No of Cities:4
 Enter the Cost Matrix:
Enter the Value of [1][1]:999
Enter the Value of [1][2]:1
Enter the Value of [1][3]:3
Enter the Value of [1][4]:6
Enter the Value of [2][1]:1
Enter the Value of [2][2]:999
Enter the Value of [2][3]:2
Enter the Value of [2][4]:3
Enter the Value of [3][1]:3
Enter the Value of [3][2]:2
Enter the Value of [3][3]:999
Enter the Value of [3][4]:1
Enter the Value of [4][1]:6
Enter the Value of [4][2]:3
Enter the Value of [4][3]:1
Enter the Value of [4][4]:999

Travelling Salesman Tour
1--->2--->4--->3--->1
Tour Cost=8_
```

RESULT:

Thus, Travelling Salesman Problem has been solved using Dynamic Programming successfully.

8 QUEEN PROBLEM USING BACKTRACKING ALGORITHM

Expt.No-11 **Date: 07-10-2022**

AIM:

To Generate all possible configurations of queens on board and print a configuration that satisfies the given constraints.

PROCEDURE:

Step-1	**Start** the Program
Step-2	**Begin** from the leftmost column
Step-3	**Check** whether all the queens are placed, return true/ print configuration
Step-4	**Check** for all rows in the current column a) if queen placed safely, mark row and column; and recursively check if we approach in the current configuration, do we obtain a solution or not b) if placing yields, a solution, return true c) if placing does not yield a solution, unmark and try other rows
Step-5	**Check** if all rows tried and solution not obtained, return false and backtrack
Step-6	**END of the Program**

PROGRAM

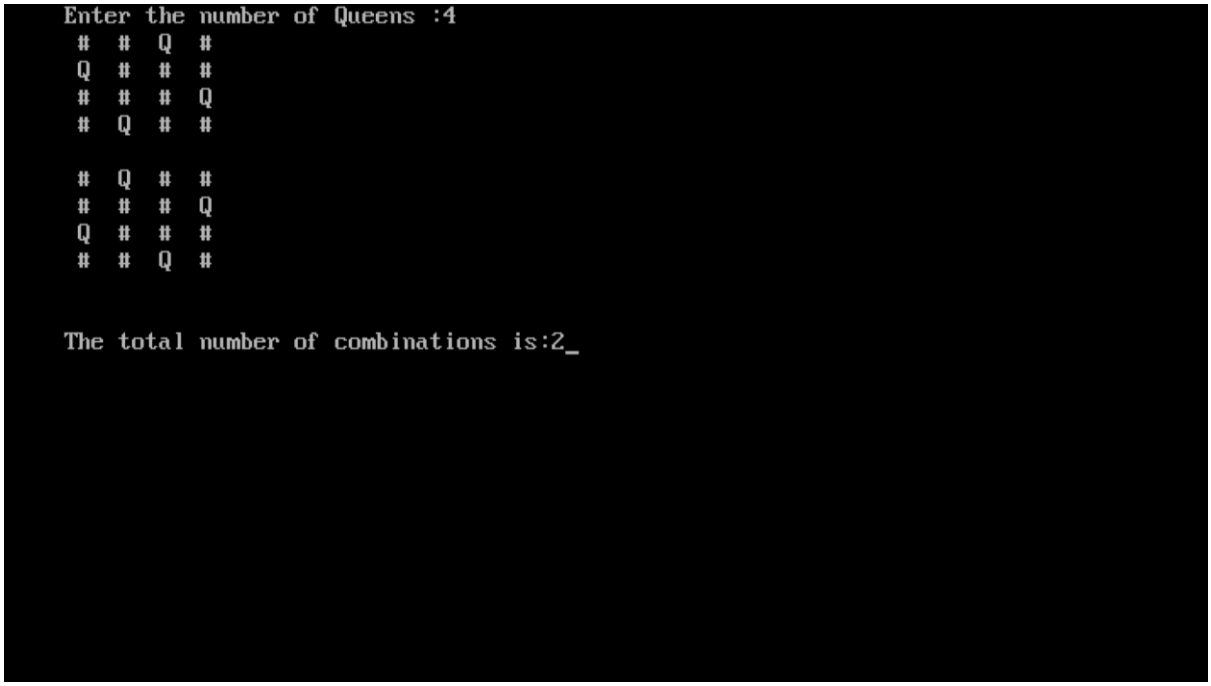
```
#include<iostream.h>
#include<conio.h>
class BackTrack
{
        int n, count, value[18][18];
public:
        BackTrack();
        void Process(int);
        void Display();
        void Refresh(int);
        void get_count()
        {
                cout<<"\nThe total number of combinations is:"<<count;
        }
};
BackTrack::BackTrack()
{
        int i,j;
        cout<<"Enter the number of Queens :";
        cin>>n;
        count=0;
        for(i=0;i<n;i++)
```

```
	{
		for(j=0;j<n;j++)
		{
			value[i][j]=0;
		}
	}
}
void main()
{
clrscr();
	BackTrack B;
	B.Process(0);
	B.get_count();
getch();
}
void BackTrack::Refresh(int position)
{
	int i;
	for(i=0; i<=n; i++)
	value[i][position] = 0;
}
void BackTrack::Display()
{
	int i,j;
	count++;
	for(i=0;i<n;i++)
	{
		for(j=0;j<n;j++)
		{
			if(value[i][j])
			{
				cout<<" Q ";
			}
			else
			{
				cout<<" # ";
			}
		}
		cout<<endl;
	}
	cout<<endl;
}
void BackTrack::Process(int position)
{
	int i,j,temp,cont;
	if(position==n)
```

```
{
        Display();
        return;
}
temp=0;
while(temp<n)
{
        cont=1;
        for(j=0; j<=n; j++)
        {
                if(value[temp][j] ==1)
                {
                        cont=0;
                }
        }
        i = temp;
        j = position;
        while(cont && (i<n) && (j<n))
        {
                if(value[i][j])
                {
                        cont=0;
                }
                i++;
                j++;
        }
        i=temp;
        j=position;
        while(cont && (i > -1) && (j > -1))
        {
                if(value[i][j])
                {
                        cont = 0;
                }
                i--;
                j--;
        }
        i =temp;
        j = position;
        while (cont &&(i>-1) && (j<n))
        {
                if(value[i][j])
                {
                        cont=0;
                }
                i--;
```

```
                j++;
            }
            i=temp;
            j=position;
            while(cont && (i<n) && (j>-1))
            {
                if(value[i][j])
                {
                    cont = 0;
                }
                i++;
                j--;
            }
            if(cont)
            {
                value[temp][position] =1;
                Process(position+1);
                Refresh(position);
            }
            temp++;
        }
}
```

SAMPLE I/O



RESULT:

Thus, 8 Queen Problem has been solved using Backtracking Algorithm successfully.