दिल्ली विश्वविद्यालय

University of Delhi

Name : Naman Jain

College roll no. : 19/1425

Exam roll no. : 19020570024

Course : B.SC.(H) CS

Semester : 6th

Computer Graphics Practical File

Question 1 : Write a program to implement Bresenham's line drawing algorithm.

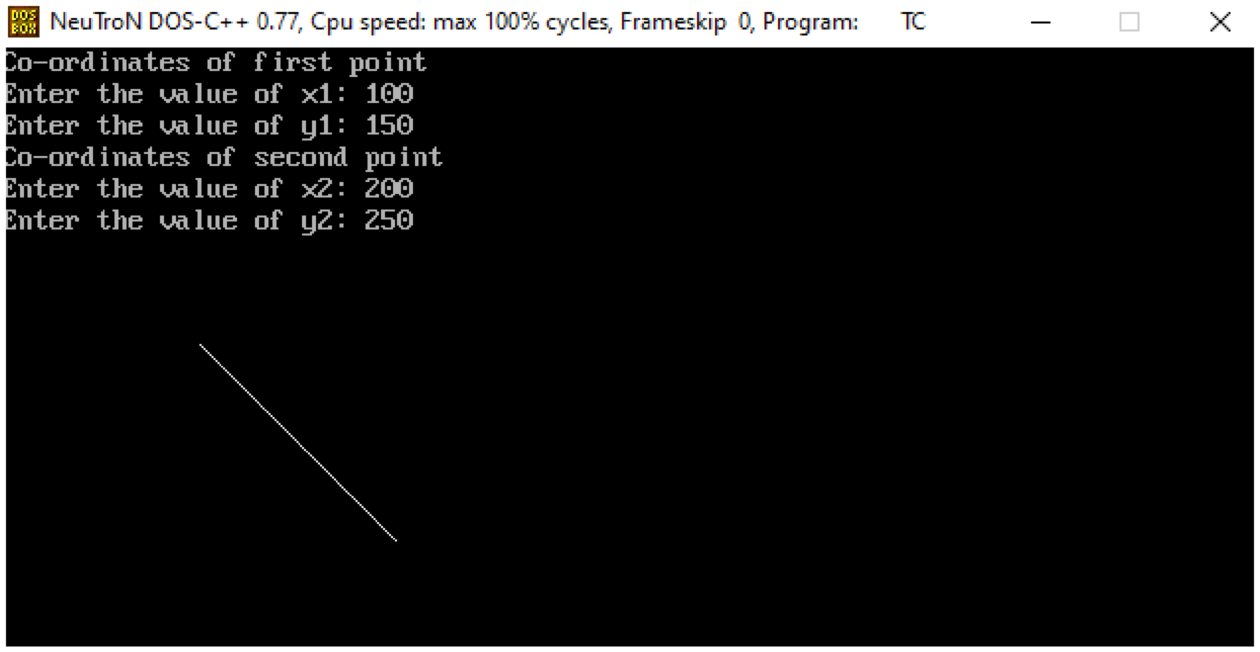
Solution :

Code -

```
#include<iostream.h>

#include<graphics.h>

#include<conio.h>

void drawline(int x0, int y0, int x1, int y1)

{

int dx, dy, p, x, y;

dx=x1-x0;

dy=y1-y0;

x=x0;

y=y0;

p=2*dy-dx;

while(x<x1)

{

if(p>=0)

{

putpixel(x,y,WHITE);

y=y+1;
p=p+2*dy-2*dx;

}

else

{

putpixel(x,y,WHITE);

p=p+2*dy;

}

x=x+1;

}

}

int main()

{

int gd=DETECT, gm;

int error, x0, y0, x1, y1;

initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

cout<<"Co-ordinates of first point"<<endl;

cout<<"Enter the value of x1: ";

cin>>x0;

cout<<"Enter the value of y1: ";

cin>>y0;

cout<<"Co-ordinates of second point"<<endl;

cout<<"Enter the value of x2: ";

cin>>x1;
```

```
cout<<"Enter the value of y2: ";

cin>>y1;

drawline(x0, y0, x1, y1);

getch();

return 0;

}
```

Output :



NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Co-ordinates of first point
Enter the value of x1: 100
Enter the value of y1: 150
Co-ordinates of second point
Enter the value of x2: 200
Enter the value of y2: 250
```

Question 2 : Write a program to implement mid-point circle drawing algorithm.

Solution :

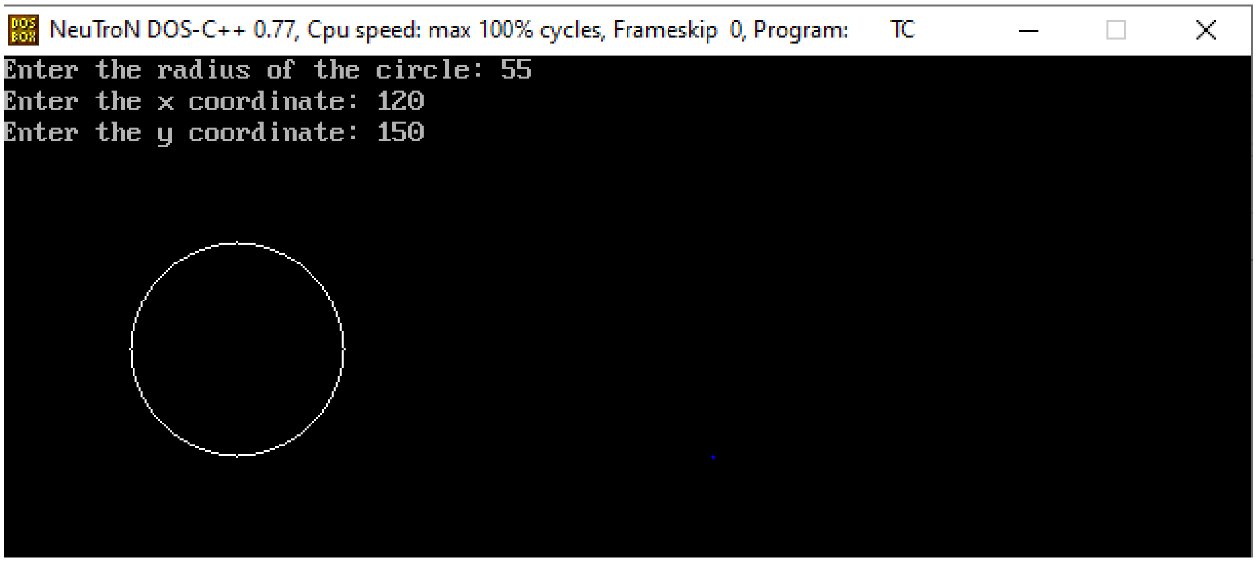
Code -

```
#include<iostream.h>
#include<graphics.h>
#include<conio.h>
void drawcircle(int x0,int y0,int r) {
```

```
int x=r;
int y=0;
int d=0;
while(x>=y) {
putpixel(x0+x,y0+y,WHITE);
putpixel(x0+y,y0+x,WHITE);
putpixel(x0
-y,y0+x,WHITE);
putpixel(x0
-x,y0+y,WHITE);
putpixel(x0
-x,y0
-y,WHITE);
putpixel(x0
-y,y0
-x,WHITE);
putpixel(x0+y,y0
-x,WHITE);
putpixel(x0+x,y0
-y,WHITE);
if(d<=0) {
y=y+1;
d+=2*y+1;
}
if(d>0) {
x=x
-1;
d
-=2*x+1;
}}}
int main() {
int d,x,y,r;
int gd=DETECT,gm;
```

```
initgraph(&gd,&gm, "C:\\TURBOC3\\BGI");
cout<<"Enter the radius of the circle: ";
cin>>r;
cout<<"Enter the x coordinate: ";
cin>>x;
cout<<"Enter the y coordinate: ";
cin>>y;
drawcircle(x,y,r);
getch();
return 0; }
```

Output :


NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the radius of the circle: 55
Enter the x coordinate: 120
Enter the y coordinate: 150
```

Question 3 : Write a program to clip a line using Cohen and Sutherland line clipping algorithm.

Solution :

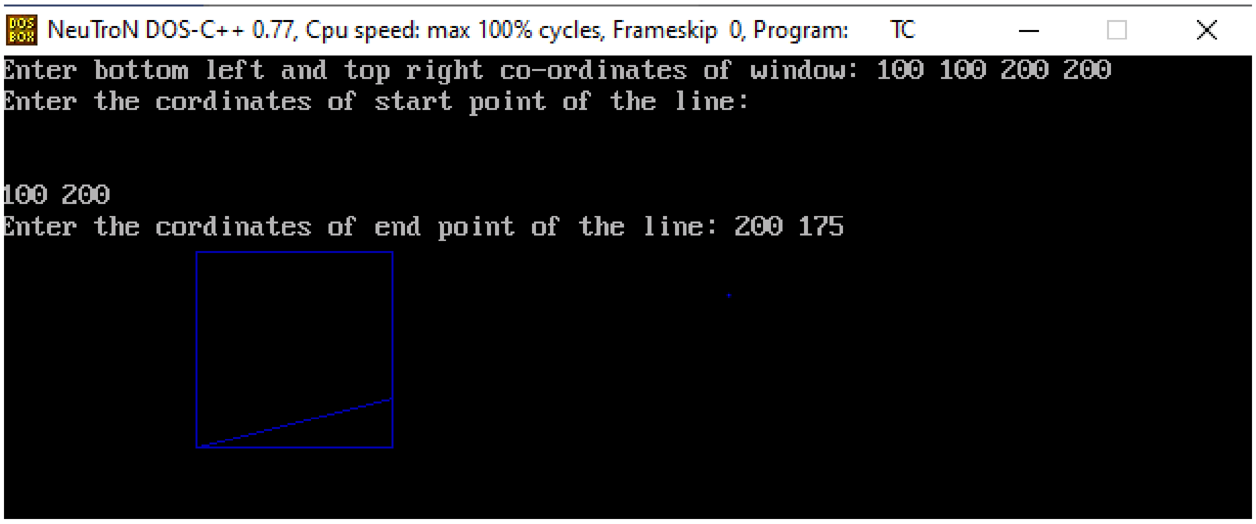
Code -

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

```
static int LEFT=1,RIGHT=2,BOTTOM=4,TOP=8,xl,yl,xh,yh;
int getcode(int x,int y)
{
int code = 0;
//Perform Bitwise OR to get outcode
if(y > yh)
code |=TOP;
if(y < yl)
code|=BOTTOM;
if(x < xl)
code |=LEFT;
if(x > xh)
code|=RIGHT;
return code;
}
int main()
{
int gdriver = DETECT,gmode;
initgraph(&gdriver,&gmode,"C:\\TURBOC3\\BGI");
setcolor(BLUE);
cout<<"Enter bottom left and top right co-ordinates of window: ";

cin>>xl>>yl>>xh>>yh;
rectangle(xl,yl,xh,yh);
int x1,y1,x2,y2;
cout<<"Enter the cordinates of start point of the line: ";
cin>>x1>>y1;
cout<<"Enter the cordinates of end point of the line: ";
cin>>x2>>y2;
line(x1,y1,x2,y2);
getch();
int outcode1=getcode(x1,y1), outcode2=getcode(x2,y2);
int accept = 0; //decides if line is to be drawn
```

```
while(1) {
float m =(float)(y2
-y1)/(x2
-x1);
//Both points inside. Accept line
if(outcode1==0 && outcode2==0) {
accept = 1;
break; }
//AND of both codes != 0.Line is outside. Reject line
else if((outcode1 & outcode2)!=0) {
break; }
else {
int x,y; int temp;
//Decide if point1 is inside, if not, calculate intersection
if(outcode1==0)
temp = outcode2;
else
temp = outcode1;
//Line clips top edge
if(temp & TOP) {
x = x1+ (yh
- y1)/m;
y = yh; }
else if(temp & BOTTOM) { //Line clips bottom edge
x = x1+ (yl
-y1)/m;
y = yl; }
else if(temp & LEFT)
{//Line clips left edge
x = xl;

y = y1+ m*(xl-x1);
}
else if(temp & RIGHT)
{ //Line clips right edge
x = xh;
y = y1+ m*(xh-x1);
}
//Check which point we had selected earlier as temp, and replace its co-ordinate s
if(temp == outcode1)
{
x1 = x; y1 = y;
outcode1 = getcode(x1,y1);
}
else
{
x2 = x;
y2= y;
outcode2 = getcode(x2,y2);
}
}
}
setcolor(WHITE);
cout<<"After clipping:";
if(accept)
line(x1,y1,x2,y2);
getch();
closegraph();
return 0;
}
```

Output :


NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter bottom left and top right co-ordinates of window: 100 100 200 200
Enter the cordinates of start point of the line:


100 200
Enter the cordinates of end point of the line: 200 175
```

Question 4 : Write a program to clip a polygon using Sutherland Hodgeman algorithm.

Solution :

Code -

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#define round(a) ((int)(a+0.5))
int k;
float xmin,ymin,xmax,ymax,arr[20],m;
void clipl(float x1,float y1,float x2,float y2)
{
if(x2-x1)
m=(y2-y1)/(x2- x1);
else
m=100000;
if(x1 >= xmin && x2 >= xmin)
{
arr[k]=x2;
arr[k+1]=y2;
```

```
k+=2;
}
if(x1 < xmin && x2 >= xmin)
{
arr[k]=xmin;
arr[k+1]=y1+m*(xmin- x1);
arr[k+2]=x2;
arr[k+3]=y2;
k+=4;
}
if(x1 >= xmin && x2 < xmin)
{
arr[k]=xmin;
arr[k+1]=y1+m*(xmin- x1);
k+=2;
}
}
void clipt(float x1,float y1,float x2,float y2)
{
if(y2-y1)
m=(x2-x1)/(y2- y1);
else
m=100000;
if(y1 <= ymax && y2 <= ymax)
{
arr[k]=x2;
arr[k+1]=y2;
k+=2;
}
if(y1 > ymax && y2 <= ymax)
 {
arr[k]=x1+m*(ymax
- y1);
```

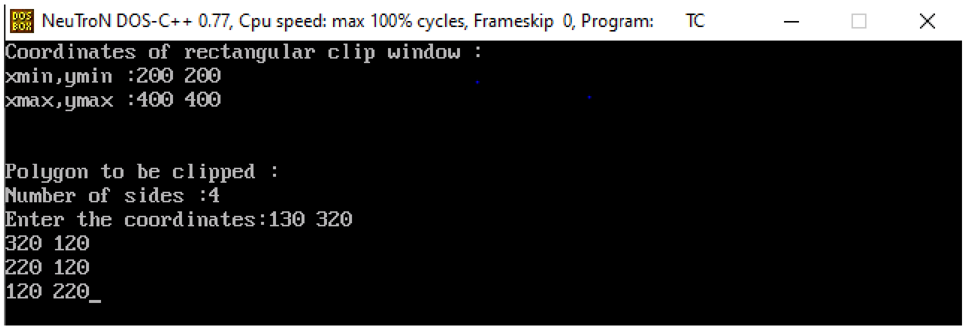
```
arr[k+1]=ymax;
arr[k+2]=x2;
arr[k+3]=y2;
k+=4; }
if(y1 <= ymax && y2 > ymax) {
arr[k]=x1+m*(ymax
- y1);
arr[k+1]=ymax;
k+=2; }}
void clipr(float x1,float y1,float x2,float y2) {
if(x2
-x1)
m=(y2
-y1)/(x2
- x1);
else
m=100000;
if(x1 <= xmax && x2 <= xmax) {
arr[k]=x2;
arr[k+1]=y2;
k+=2; }
if(x1 > xmax && x2 <= xmax) {
arr[k]=xmax;
arr[k+1]=y1+m*(xmax
- x1);
arr[k+2]=x2;
arr[k+3]=y2;
k+=4; }
if(x1 <= xmax && x2 > xmax) {
arr[k]=xmax;
arr[k+1]=y1+m*(xmax
- x1);
k+=2; }}
```

```
void clipb(float x1,float y1,float x2,float y2) {
if(y2
-y1)
m=(x2
-x1)/(y2
- y1);
else
m=100000;

if(y1 >= ymin && y2 >= ymin)
{
arr[k]=x2;
arr[k+1]=y2;
k+=2;
}
if(y1 < ymin && y2 >= ymin)
{
arr[k]=x1+m*(ymin- y1);
arr[k+1]=ymin;
arr[k+2]=x2;
arr[k+3]=y2;
k+=4;
}
if(y1 >= ymin && y2 < ymin)
{
arr[k]=x1+m*(ymin- y1);
arr[k+1]=ymin;
k+=2;
}
}
int main()
{
int gdriver=DETECT,gmode,n,poly[20];
```

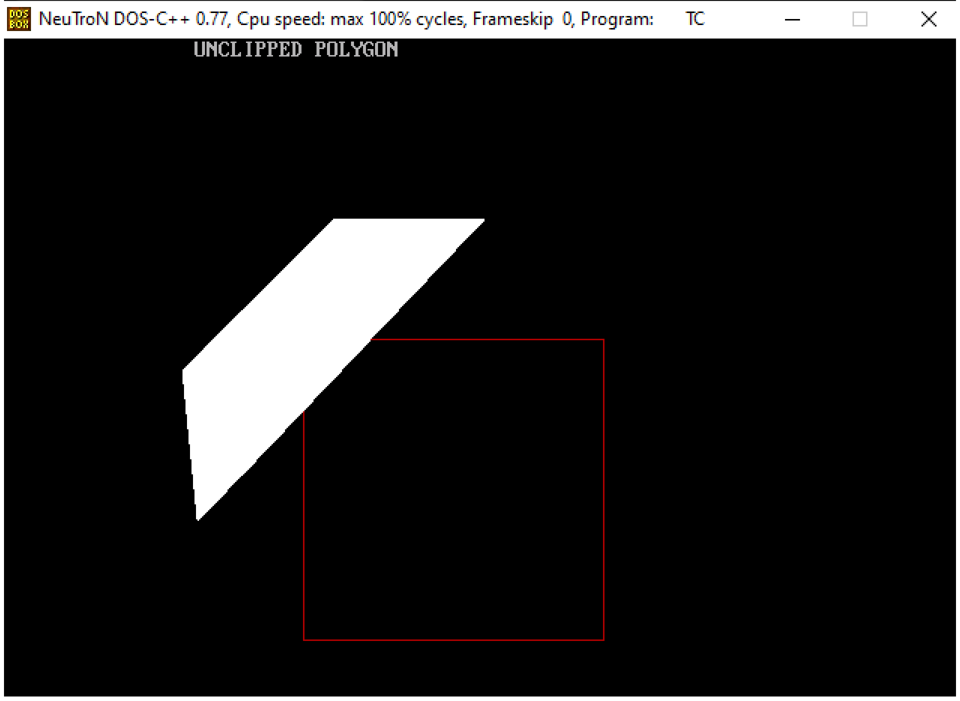
```
float xi,yi,xf,yf,polyy[20];

cout<<"Coordinates of rectangular clip window :\nxmin,ymin :";
cin>>xmin>>ymin;
cout<<"xmax,ymax :";
cin>>xmax>>ymax;
cout<<"\n\nPolygon to be clipped :\nNumber of sides :";
cin>>n;
cout<<"Enter the coordinates:";
for(int i=0;i<2*n;i++)
cin>>polyy[i];
polyy[i]=polyy[0];
polyy[i+1]=polyy[1];
for(i=0;i < 2*n+2;i++)
poly[i]=round(polyy[i]);
initgraph(&gdriver,&gmode,"C:\\TURBOC3\\BGI");
setcolor(RED);
rectangle(xmin,ymax,xmax,ymin);
cout<<"\t\tUNCLIPPED POLYGON";
setcolor(WHITE);
fillpoly(n,poly);
getch();
cleardevice();
k=0;

for(i=0;i < 2*n;i+=2)
clipl(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
n=k/2;
for(i=0;i < k;i++)
polyy[i]=arr[i];
polyy[i]=polyy[0];
polyy[i+1]=polyy[1];
k=0;
```

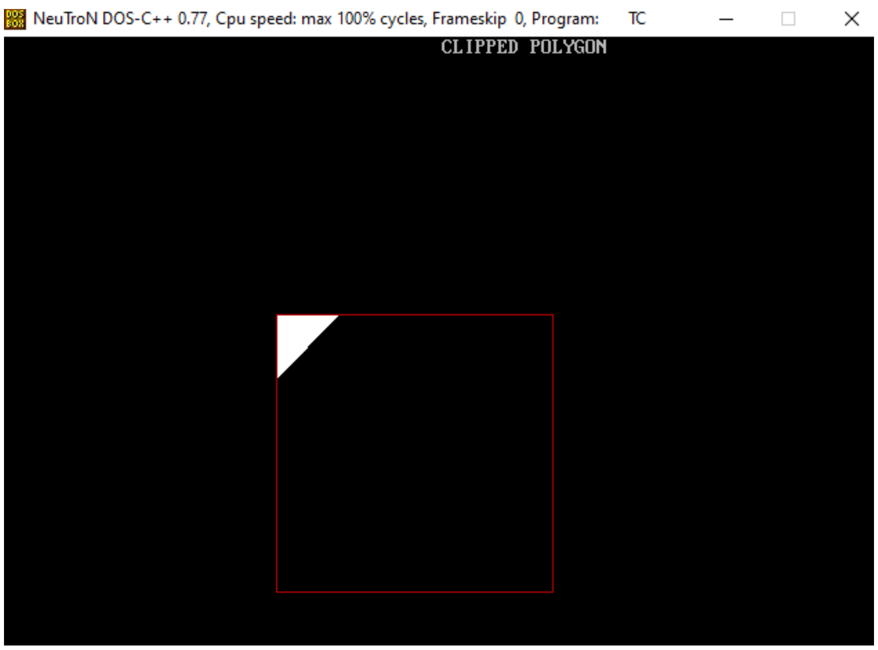
```
for(i=0;i < 2*n;i+=2)
clipt(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
n=k/2;
for(i=0;i < k;i++)
polyy[i]=arr[i];
polyy[i]=polyy[0];
polyy[i+1]=polyy[1];
k=0;
for(i=0;i < 2*n;i+=2)
clipr(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
n=k/2;
for(i=0;i < k;i++)
polyy[i]=arr[i];
polyy[i]=polyy[0];
polyy[i+1]=polyy[1];
k=0;
for(i=0;i < 2*n;i+=2)
clipb(polyy[i],polyy[i+1],polyy[i+2],polyy[i+ 3]);
for(i=0;i < k;i++)
poly[i]=round(arr[i]);
if(k)
fillpoly(k/2,poly);
setcolor(RED);
rectangle(xmin,ymax,xmax,ymin);
cout<<"\tCLIPPED POLYGON";
getch();
closegraph();
return 0;
 }
```

Output :



```
Coordinates of rectangular clip window :
xmin,ymin :200 200
xmax,ymax :400 400


Polygon to be clipped :
Number of sides :4
Enter the coordinates:130 320
320 120
220 120
120 220_
```





Question 5 : Write a program to fill a polygon using Scan line fill algorithm.

Solution :

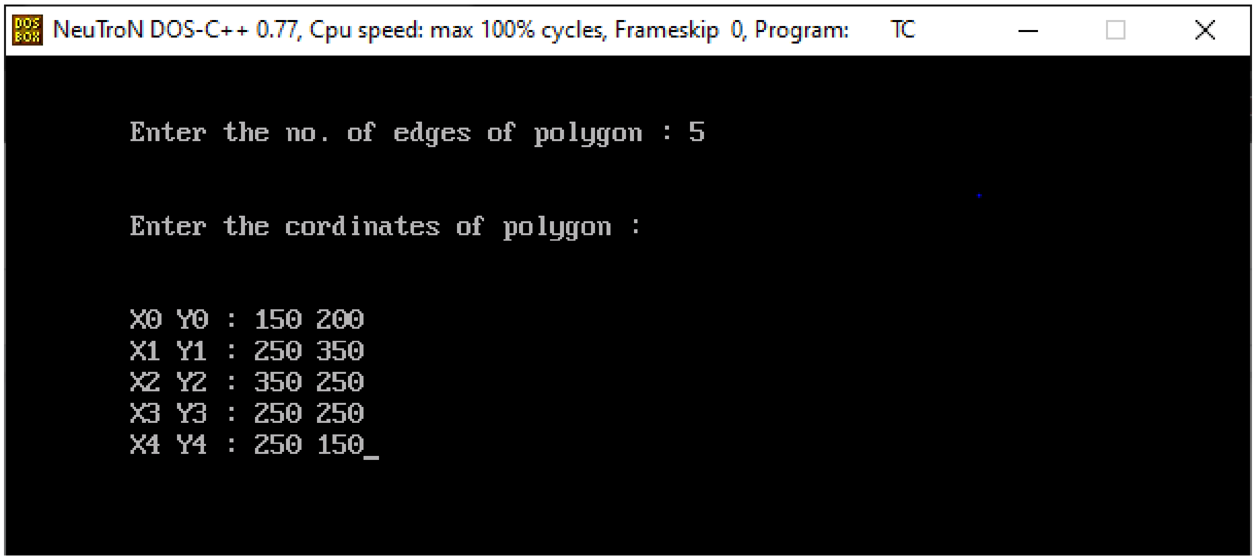
Code -

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
int main()
{
int n,i,j,k,gd,gm,dy,dx;
int x,y,temp;
int a[20][2],xi[20];
float slope[20];

printf("\n\n\tEnter the no. of edges of polygon : ");
scanf("%d",&n);
printf("\n\n\tEnter the cordinates of polygon :\n\n\n ");
for(i=0;i<n;i++)
{
printf("\tX%d Y%d : ",i,i);
scanf("%d %d",&a[i][0],&a[i][1]);
}

a[n][0]=a[0][0];
a[n][1]=a[0][1];
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
for(i=0;i<n;i++) {
line(a[i][0],a[i][1],a[i+1][0],a[i+1][1]); }
getch();
for(i=0;i<n;i++) {
```

```
dy=a[i+1][1]
-a[i][1];
dx=a[i+1][0]
-a[i][0];
if(dy==0) slope[i]=1.0;
if(dx==0) slope[i]=0.0;
if((dy!=0)&&(dx!=0)) {
slope[i]=(float) dx/dy; }}
for(y=0;y< 480;y++) {
k=0;
for(i=0;i<n;i++) {
if( ((a[i][1]<=y)&&(a[i+1][1]>y))||
((a[i][1]>y)&&(a[i+1][1]<=y))) {
xi[k]=(int)(a[i][0]+slope[i]*(y
-a[i][1]));
k++; }}
for(j=0;j<k
-1;j++)
for(i=0;i<k
-1;i++)
{
if(xi[i]>xi[i+1]) {
temp=xi[i];
xi[i]=xi[i+1];
xi[i+1]=temp; }}

setcolor(35);
for(i=0;i<k;i+=2)
{
line(xi[i],y,xi[i+1]+1,y);
getch();
}
}
```

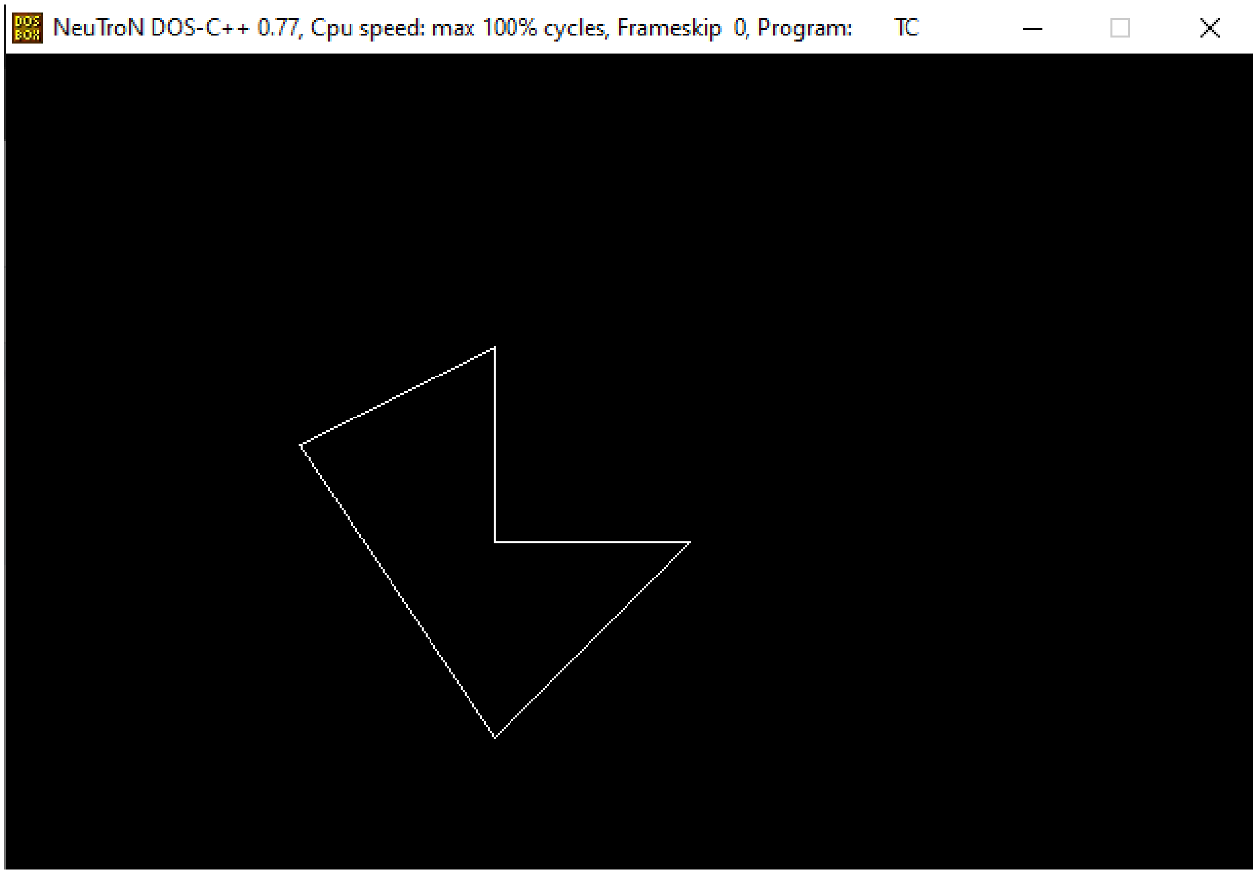
```
return 0;
}
```

Output :



NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
Enter the no. of edges of polygon : 5


Enter the cordinates of polygon :


X0 Y0 : 150 200
X1 Y1 : 250 350
X2 Y2 : 350 250
X3 Y3 : 250 250
X4 Y4 : 250 150_
```



NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

Question 6 : Write a program to apply various 2D transformations on a 2D object (use homogenous 64 Coordinates).

Solution :

Code -

```
#include<graphics.h>
#include<stdlib.h>
#include<stdio.h>
#include<iostream.h>
#include<conio.h>
#include<math.h>

int mat[3][3];

//DDA line function
void dda_line(int x1 , int y1 , int x2 , int y2 , int col)
{
int dx , dy , st;
dx = x2 - x1;
dy = y2 - y1;
float y , x , xinc , yinc;
int xmid , ymid;
xmid = getmaxx()/2;
ymid = getmaxy()/2;
if(abs(dx) > abs(dy))
{
st = abs(dx);
}
else
{
st = abs(dy);
}

xinc = dx / st;
yinc = dy / st;
x = x1;
```

```
y = y1;
for(int i=0 ; i<st ; i++)
{
x += xinc;
y += yinc;
putpixel(ceil(x) + xmid , ymid - ceil(y),col);
}
}
//Rotation
void rotate()
{
int xmid , ymid;
xmid = getmaxx()/2;
ymid = getmaxy()/2;
line(xmid , 0 , xmid , getmaxy());
line(0 , ymid , getmaxx() , ymid);
int c[3][2] ,l , m, i , j , k;
int a[3][2]={{200,200},{200,100},{100,200}};
int t[2][2]={{0,1},{-1,0}};
for( i = 0 ; i < 3 ; i++)
{
for(j=0 ; j<2 ; j++)
{
c[i][j]=0;
}
}
//Original Triangle
dda_line(a[0][0],a[0][1],a[1][0],a[1][1],YELLOW);
dda_line(a[1][0],a[1][1],a[2][0],a[2][1],YELLOW);
dda_line(a[2][0],a[2][1],a[0][0],a[0][1],YELLOW);
for ( i=0;i<3;i++)
{
for ( j=0;j<2;j++)
{
for ( k=0;k<2;k++)
{
c[i][j]=c[i][j]+(a[i][k]*t[k][j]);
}
}
}
```

```
//Transformed Triangle
dda_line(c[0][0],c[0][1],c[1][0],c[1][1],GREEN);
dda_line(c[1][0],c[1][1],c[2][0],c[2][1],GREEN);
dda_line(c[2][0],c[2][1],c[0][0],c[0][1],GREEN);
}

//Reflection
 void reflection()
{
int xmid , ymid;
xmid = getmaxx()/2;
ymid = getmaxy()/2;
line(xmid , 0 , xmid , getmaxy());
line(0 , ymid , getmaxx() , ymid);
int c[3][2] ,l , m, i , j , k;
int a[3][2]={{200,200},{200,100},{100,200}};
int t[2][2]={{0,-1},{-1,0}};
for( i = 0 ; i < 3 ; i++)
{
for(j=0 ; j<2 ; j++)
{
c[i][j]=0;
}
}
//Original Triangle
dda_line (a[0][0],a[0][1],a[1][0],a[1][1],YELLOW);
dda_line(a[1][0],a[1][1],a[2][0],a[2][1],YELLOW);
dda_line(a[2][0],a[2][1],a[0][0],a[0][1],YELLOW);
for ( i=0;i<3;i++)
{
for ( j=0;j<2;j++)
{
for ( k=0;k<2;k++)
{
c[i][j]=c[i][j]+(a[i][k]*t[k][j]);
}
}
}
//Transformed Triangle
dda_line(c[0][0],c[0][1],c[1][0],c[1][1],GREEN);
dda_line(c[1][0],c[1][1],c[2][0],c[2][1],GREEN);
dda_line(c[2][0],c[2][1],c[0][0],c[0][1],GREEN);
}

//Scaling
void scaling()
{
int xmid , ymid;
xmid = getmaxx()/2;
ymid = getmaxy()/2;
line(xmid , 0 , xmid , getmaxy());
line(0 , ymid , getmaxx() , ymid);

int c[3][2] ,l , m, i , j , k;
int a[3][2]={{20,20},{20,10},{10,20}};
int t[2][2]={{5,0},{0,5}};
for( i = 0 ; i < 3 ; i++)
{
for(j=0 ; j<2 ; j++)
{ c[i][j]=0;
}
}
//Original Triangle
dda_line(a[0][0],a[0][1],a[1][0],a[1][1],YELLOW);
dda_line(a[1][0],a[1][1],a[2][0],a[2][1],YELLOW);
dda_line(a[2][0],a[2][1],a[0][0],a[0][1],YELLOW);
for ( i=0;i<3;i++)
{
for ( j=0;j<2;j++)
{
for ( k=0;k<2;k++)
{
c[i][j]=c[i][j]+(a[i][k]*t[k][j]);
}
}
}
//Transformed Triangle
dda_line(c[0][0],c[0][1],c[1][0],c[1][1],GREEN);
dda_line(c[1][0],c[1][1],c[2][0],c[2][1],GREEN);
dda_line(c[2][0],c[2][1],c[0][0],c[0][1],GREEN);
```

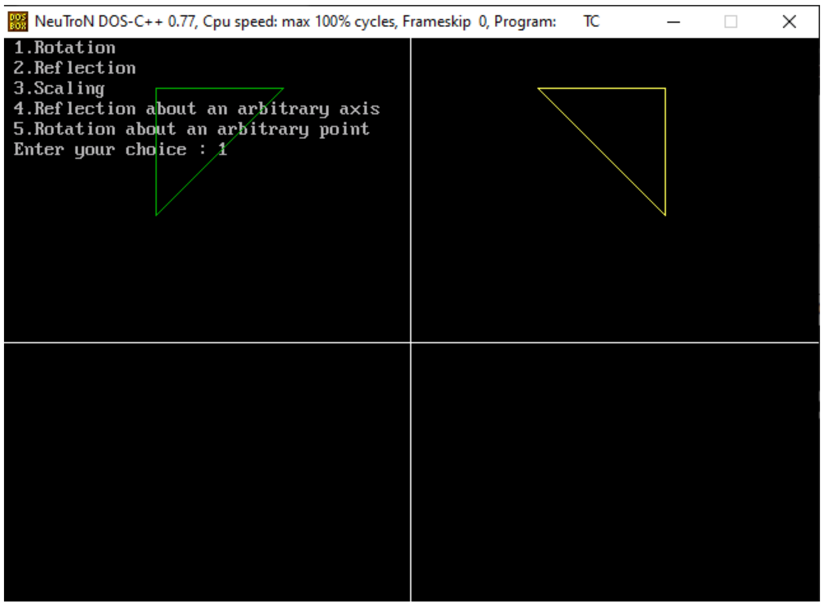
```
}

void multi(int a[3][3] , int b[3][3] )
{
int i , j ,k; int c[3][3];
for( i = 0 ; i < 3 ; i++)
{
for(j=0 ; j< 3 ; j++)
{
c[i][j]=0;
}
}
for ( i=0;i<3;i++)
{
for ( j=0;j<3;j++)
{
for ( k=0;k<3;k++)
{
c[i][j]=c[i][j]+(a[i][k]*b[k][j]);
}
}
}
for( i = 0 ; i < 3 ; i++)
{
for(j=0 ; j< 3 ; j++)
{
mat[i][j]=c[i][j];
}
}
}

//Reflection About an arbitrary line
void reflection_arbitrary()
{
int xmid , ymid;
xmid = getmaxx()/2;
ymid = getmaxy()/2;
line(xmid , 0 , xmid , getmaxy());

line(0 , ymid , getmaxx() , ymid);
```

```
int a[3][3]={{200,200,1},{200,100,1},{100,200,1}};
int t[3][3]={{1,0,0},{0,1,0},{0,0,1}};
int r[3][3]={{-1,0,0},{0,-1,0},{0,0,1}};
int ref[3][3]={{1,0,0},{0,-1,0},{0,0,1}};
int rinv[3][3]={{-1,0,0},{0,-1,0},{0,0,1}};
int tinv[3][3]={{1,0,0},{0,1,0},{0,1,1}};
//Original Triangle
dda_line(a[0][0],a[0][1],a[1][0],a[1][1],YELLOW);
dda_line(a[1][0],a[1][1],a[2][0],a[2][1],YELLOW);
dda_line(a[2][0],a[2][1],a[0][0],a[0][1],YELLOW);
multi(t,r); multi(mat,ref); multi(mat,rinv); multi(mat,tinv); multi(a,mat);
//Transformed Triangle
dda_line(mat[0][0],mat[0][1],mat[1][0],mat[1][1],GREEN);
dda_line(mat[1][0],mat[1][1],mat[2][0],mat[2][1],GREEN);
dda_line(mat[2][0],mat[2][1],mat[0][0],mat[0][1],GREEN);
}

//Rotation About an arbitrary point
void rotation_arbitrary()
{
int xmid , ymid;
xmid = getmaxx()/2;
ymid = getmaxy()/2;
line(xmid , 0 , xmid , getmaxy());
line(0 , ymid , getmaxx() , ymid);
int c[3][3] , i , j , k;
int l[1][3]={{200,200,1}};
int a[3][3]={{200,200,1},{200,100,1},{100,200,1}};
int t[3][3]={{1,0,0},{0,1,0},{-133,-133,1}};
int r[3][3]={{-1,0,0},{0,-1,0},{0,0,1}};
int tinv[3][3]={{1,0,0},{0,1,0},{133,133,1}};
//Original Triangle
dda_line(a[0][0],a[0][1],a[1][0],a[1][1],YELLOW);
dda_line(a[1][0],a[1][1],a[2][0],a[2][1],YELLOW);
dda_line(a[2][0],a[2][1],a[0][0],a[0][1],YELLOW);
multi(t,r);
multi(mat,tinv);
for( i = 0 ; i < 3 ; i++)
{
for(j=0 ; j<3 ; j++)
```

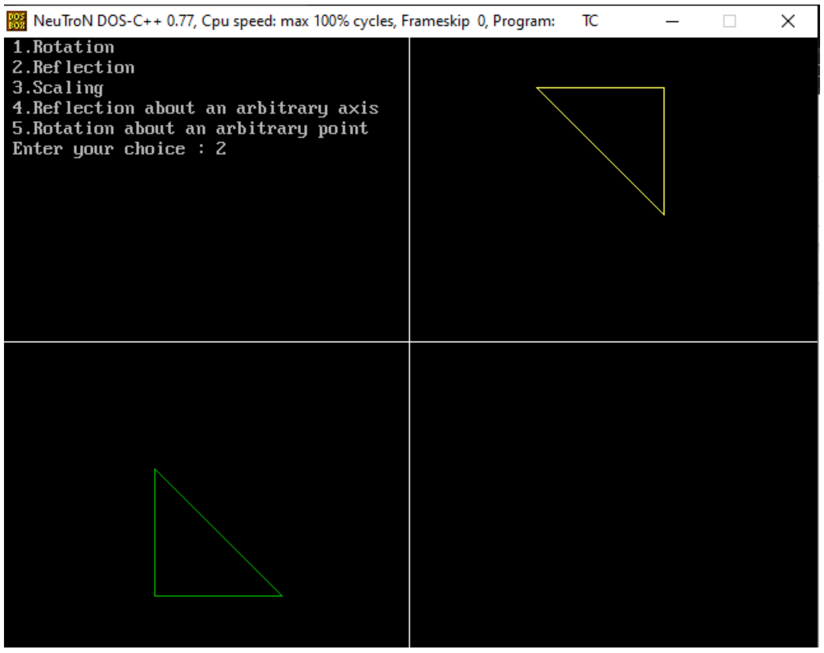
```
{
c[i][j]=0;
}
}
for ( i=0;i<3;i++)
{
for ( j=0;j<3;j++)
{
for ( k=0;k<3;k++)
{
c[i][j]=c[i][j]+(a[i][k]*mat[k][j]);
}
}
}
//Transformed Triangle
dda_line(c[0][0],c[0][1],c[1][0],c[1][1],GREEN);
dda_line(c[1][0],c[1][1],c[2][0],c[2][1],GREEN);
dda_line(c[2][0],c[2][1],c[0][0],c[0][1],GREEN);
}

//main function
int main()
{
int gdriver = DETECT , gmode , errorcode;
initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI");
int n , m;
cout<<" 1.Rotation \n 2.Reflection \n 3.Scaling \n 4.Reflection about an arbitrary axis \n"; cout<<"
5.Rotation about an arbitrary point\n";
cout<<" Enter your choice : "; cin>>n;
switch(n)
{
case 1 : rotate();
 break;
case 2 : reflection();
 break;
case 3 : scaling();
break;
case 4 : reflection_arbitrary();
 break;
case 5 : rotation_arbitrary();
```

```
 break;
default : cout<<"Invalid Choice\n";
}
getch();
return 0;
}
```

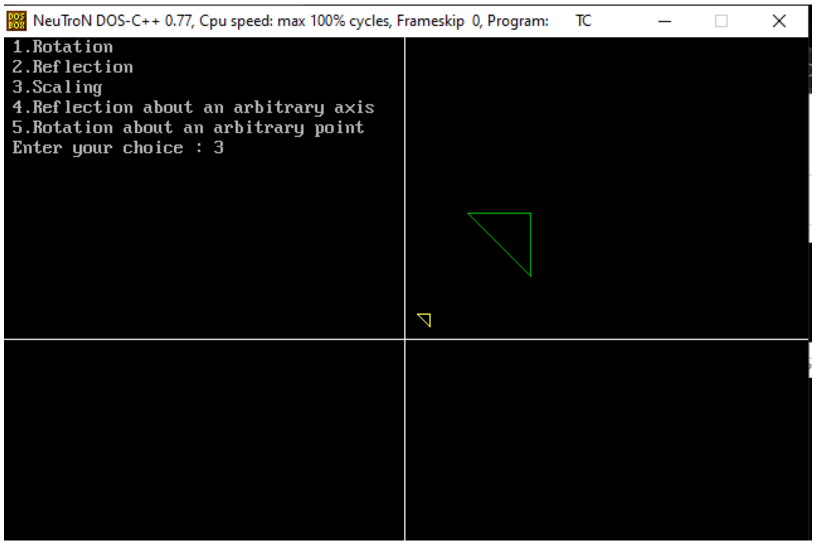
Output :



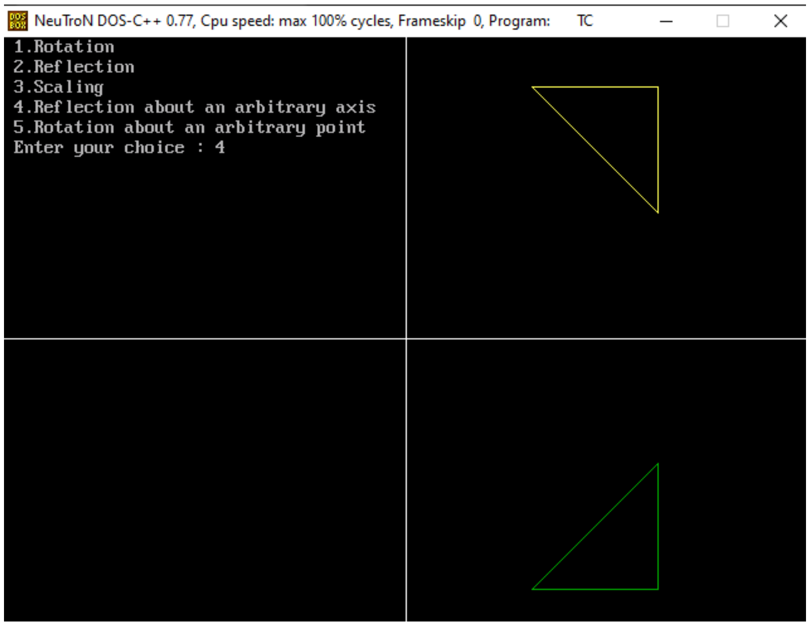
```
1.Rotation
2.Reflection
3.Scaling
4.Reflection about an arbitrary axis
5.Rotation about an arbitrary point
Enter your choice : 1
```



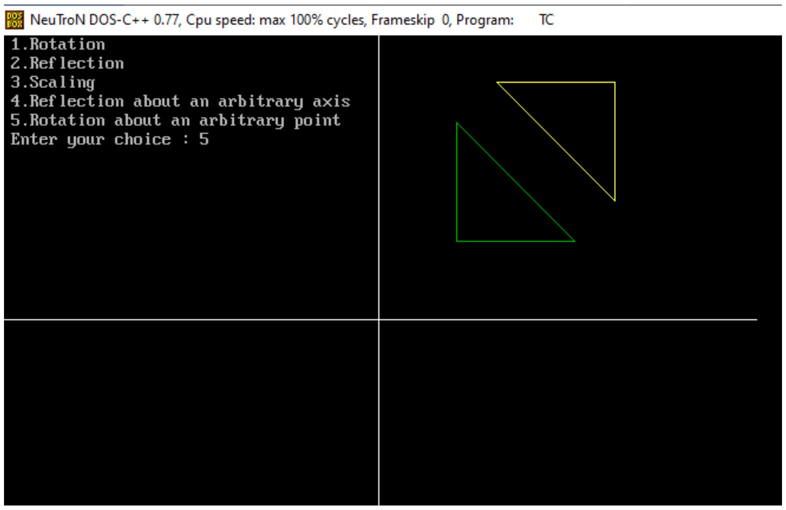
```
1.Rotation
2.Reflection
3.Scaling
4.Reflection about an arbitrary axis
5.Rotation about an arbitrary point
Enter your choice : 2
```



```
NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip  0, Program:     TC
1.Rotation
2.Reflection
3.Scaling
4.Reflection about an arbitrary axis
5.Rotation about an arbitrary point
Enter your choice : 3
```



```
NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip  0, Program:     TC
1.Rotation
2.Reflection
3.Scaling
4.Reflection about an arbitrary axis
5.Rotation about an arbitrary point
Enter your choice : 4
```



```
NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip  0, Program:     TC
1.Rotation
2.Reflection
3.Scaling
4.Reflection about an arbitrary axis
5.Rotation about an arbitrary point
Enter your choice : 5
```

Question 7 : Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it.

Solution :

Code -

```
#include<iostream.h>
#include<dos.h>
#include<stdio.h>
#include<math.h>
#include<conio.h>
#include<graphics.h>
#include<process.h>


int gd=DETECT,gm;
double x,x2,y,y2;

//Creating draw cube function for drawing cube
void draw_cube(double edge[20][3])
{
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
int i;
clearviewport();
for(i=0;i < 19;i++)
{
x=edge[i][0]+edge[i][2]*(cos(2.3562));
y=edge[i][1]-edge[i][2]*(sin(2.3562));
x2=edge[i+1][0]+edge[i+1][2]*(cos(2.3562));
y2=edge[i+1][1]-edge[i+1][2]*(sin(2.3562));
line(x+320,240-y,x2+320,240-y2);
}
```

```
line(320,240,320,25);
line(320,240,550,240);
line(320,240,150,410);
getch();
closegraph();
}
//Scaling Function
void scale(double edge[20][3])
{
double a,b,c; int i;
cout<<"Enter The Scaling Factors "<<endl;
cin>>a>>b>>c;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
clearviewport();
for(i=0;i < 20;i++)
{ // Scaling Factors a, b, c at X, Y, Z
edge[i][0]=edge[i][0]*a;
edge[i][1]=edge[i][1]*b;
edge[i][2]=edge[i][2]*c;
}
draw_cube(edge);
closegraph();
}
// Creating Translation function
void translate(double edge[20][3])
{
int a,b,c;
int i;
cout<<"Enter The Translation Factors"<<endl;
cin>>a>>b>>c;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
clearviewport();
for(i=0;i < 20;i++)
```

```
{
//Three Translation Factors a, b, c
edge[i][0]+=a;
edge[i][0]+=b;
edge[i][0]+=c;
}
draw_cube(edge);
closegraph();
}

// Creating Rotation About an Axes function
void rotate(double edge[20][3])
{
int ch;
int i;
double temp,theta,temp1;

cout<<"Rotation About"<<endl;
cout<<"1 X-Axis "<<endl;
cout<<"2 Y-Axis"<<endl;
cout<<"3 Z-Axis "<<endl;
cout<<"Enter Your Choice "<<endl;
cin>>ch;
switch(ch)
{ //For X-axis
case 1:
cout<<" Enter The Angle ";
cin>>theta;
theta=(theta*3.14)/180;
for(i=0;i < 20;i++)
{
edge[i][0]=edge[i][0];
temp=edge[i][1];
```

```
temp1=edge[i][2];
//Transformation Matrix For X-axis
edge[i][1]=temp*cos(theta)-temp1*sin(theta);
edge[i][2]=temp*sin(theta)+temp1*cos(theta);
}
draw_cube(edge);
break;
//For Y-axis
case 2:
cout<<" Enter The Angle ";
cin>>theta;
theta=(theta*3.14)/180;
for(i=0;i < 20;i++)
{
edge[i][1]=edge[i][1];
temp=edge[i][0];
temp1=edge[i][2];
//Transformation Matrix For Y-axis
edge[i][0]=temp*cos(theta)+temp1*sin(theta);
edge[i][2]=-temp*sin(theta)+temp1*cos(theta);
}
draw_cube(edge);
break;
//For Z-axis
case 3:
cout<<" Enter The Angle ";
cin>>theta;
theta=(theta*3.14)/180;
for(i=0;i < 20;i++)
{
edge[i][2]=edge[i][2];
temp=edge[i][0];
temp1=edge[i][1];
```

```
//Transformation Matrix For Z-axis
edge[i][0]=temp*cos(theta)-temp1*sin(theta);
edge[i][1]=temp*sin(theta)+temp1*cos(theta);

}
draw_cube(edge);
break;
}
}
// Creating Reflection About an Axes function
void reflect(double edge[20][3])
{
int ch; int i;

cout<<"Reflection About "<<endl;
cout<<"1 X-Axis"<<endl;
cout<<"2 Y-Axis "<<endl;
cout<<"3 Z-Axis "<<endl;
cout<<"Enter Your Choice "<<endl;
cin>>ch;
switch(ch)
{ //For X-axis
case 1:
for(i=0;i < 20;i++)
{
edge[i][0]=edge[i][0];
edge[i][1]=-edge[i][1];
edge[i][2]=-edge[i][2];
}
draw_cube(edge);
break;
//For Y-axis
case 2:
```

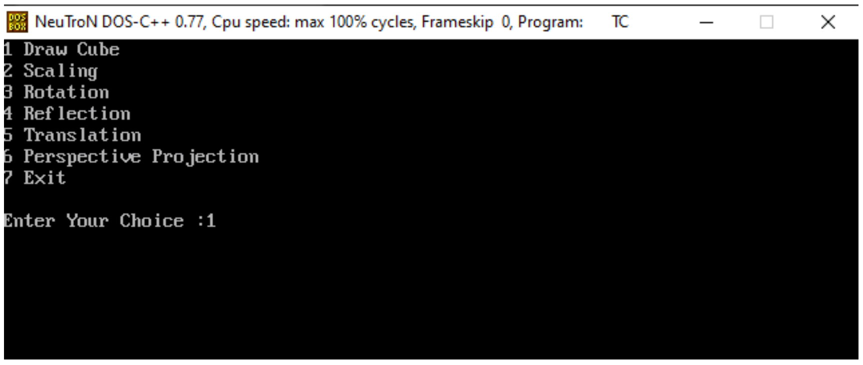
```
for(i=0;i < 20;i++)
{
edge[i][1]=edge[i][1];
edge[i][0]=-edge[i][0];
edge[i][2]=-edge[i][2];
}
draw_cube(edge);
break;
//For Z-axis
case 3:
for(i=0;i < 20;i++)
{
edge[i][2]=edge[i][2];
edge[i][0]=-edge[i][0];
edge[i][1]=-edge[i][1];
}
draw_cube(edge);
break;
}
}
// Creating Perspective Projection About an Axes function
void perspect(double edge[20][3])
{
int ch; int i;
double p,q,r;
cout<<"Perspective Projection About"<<endl;
cout<<"1 X-Axis "<<endl;
cout<<"2 Y-Axis "<<endl;
cout<<"3 Z-Axis"<<endl;
cout<<"Enter Your Choice :"<<endl;
cin>>ch;
switch(ch)
{
```

```
//For X-axis
case 1:
cout<<" Enter P :";
cin>>p;
for(i=0;i < 20;i++)
{
edge[i][0]=edge[i][0]/(p*edge[i][0]+1);
edge[i][1]=edge[i][1]/(p*edge[i][0]+1);
edge[i][2]=edge[i][2]/(p*edge[i][0]+1);
}
draw_cube(edge);
break;
//For Y-axis
case 2: cout<<" Enter Q :";
cin>>q;
for(i=0;i < 20;i++)
{
edge[i][1]=edge[i][1]/(edge[i][1]*q+1);
edge[i][0]=edge[i][0]/(edge[i][1]*q+1);
edge[i][2]=edge[i][2]/(edge[i][1]*q+1);
}
draw_cube(edge);
break;
//For Z-axis
case 3:
cout<<" Enter R :";
cin>>r;
for(i=0;i < 20;i++)
{
edge[i][2]=edge[i][2]/(edge[i][2]*r+1);
edge[i][0]=edge[i][0]/(edge[i][2]*r+1);
edge[i][1]=edge[i][1]/(edge[i][2]*r+1);
}
```

```
draw_cube(edge);
break;
}
closegraph();
}
//Main Function
int main()
{
int choice;
double edge[20][3]={
100,0,0,
100,100,0,
0,100,0,
0,100,100,
0,0,100,
0,0,0,
100,0,0,
100,0,100,
100,75,100,
75,100,100,
100,100,75,
100,100,0,
100,100,75,
100,75,100,
75,100,100,
0,100,100,
0,100,0,
0,0,0,
0,0,100,
100,0,100
};
while(1)
{
```

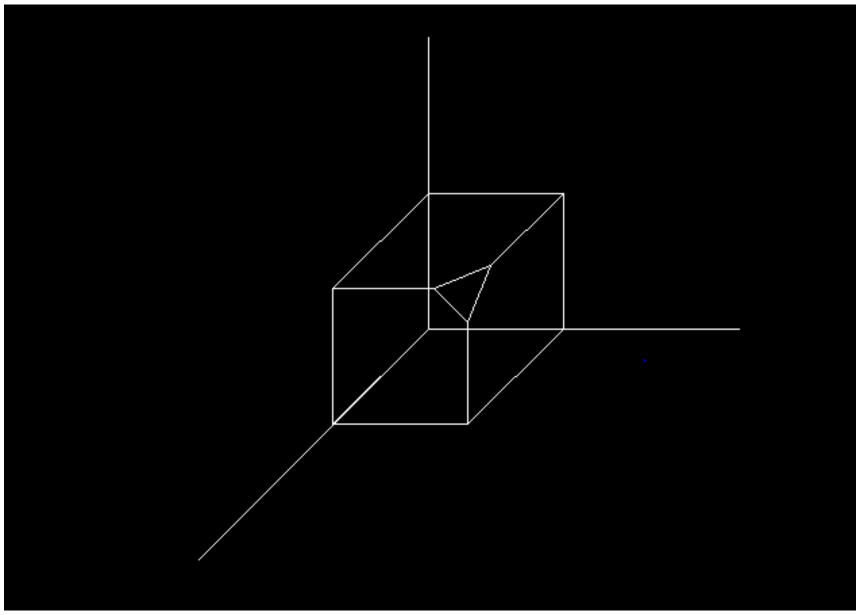
```
cout<<"1 Draw Cube "<<endl;
cout<<"2 Scaling "<<endl;
cout<<"3 Rotation "<<endl;
cout<<"4 Reflection "<<endl;
cout<<"5 Translation "<<endl;
cout<<"6 Perspective Projection "<<endl;
cout<<"7 Exit "<<endl;
cout<<"\nEnter Your Choice :";
cin>>choice;
switch(choice)
{
case 1:
draw_cube(edge);
break;
case 2:
scale(edge);
break;

case 3:
rotate(edge);
break;

case 4:
reflect(edge);
break;

case 5:
translate(edge);
break;

case 6:
perspect(edge);
break;

case 7:
exit(0);

default:
cout<<" Press A Valid Key...!!! ";
getch();
break;
}
closegraph();
}
return 0;
}
```

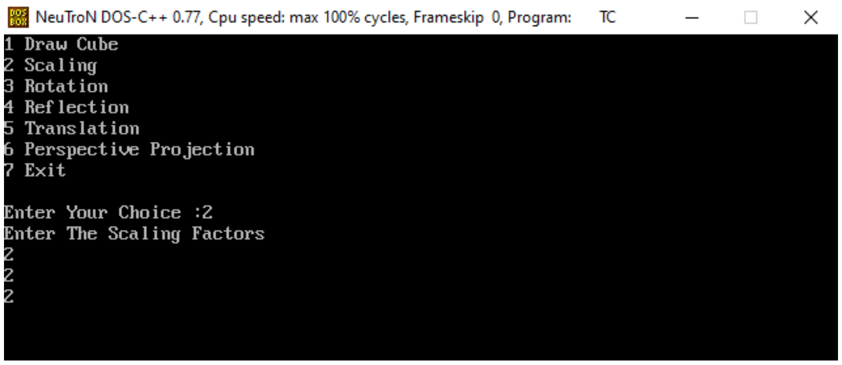
Output :



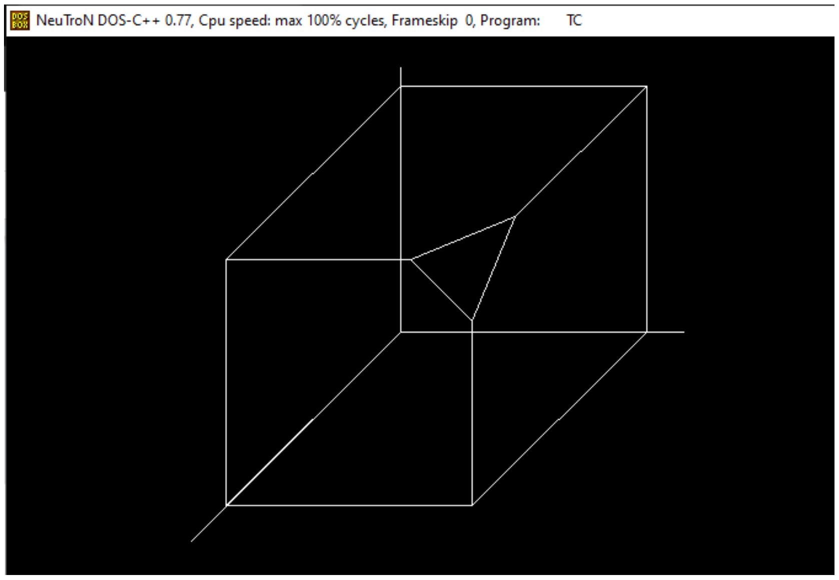
NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

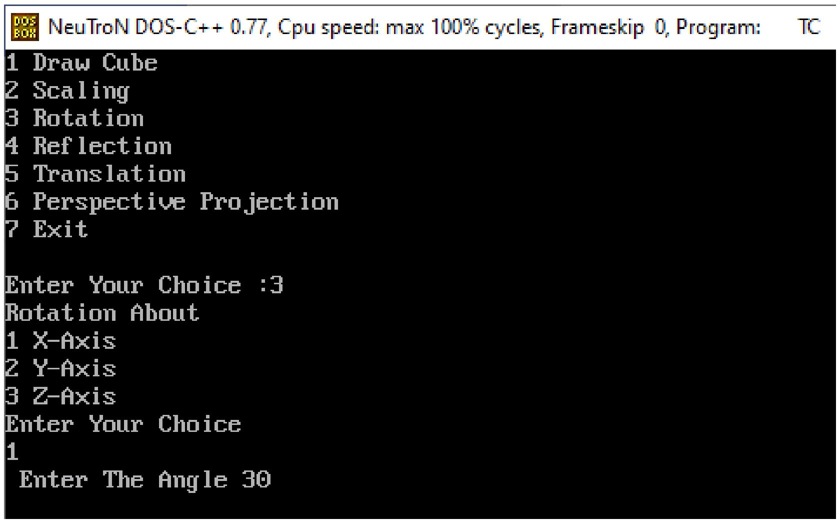
```
1 Draw Cube
2 Scaling
3 Rotation
4 Reflection
5 Translation
6 Perspective Projection
7 Exit

Enter Your Choice :1
```



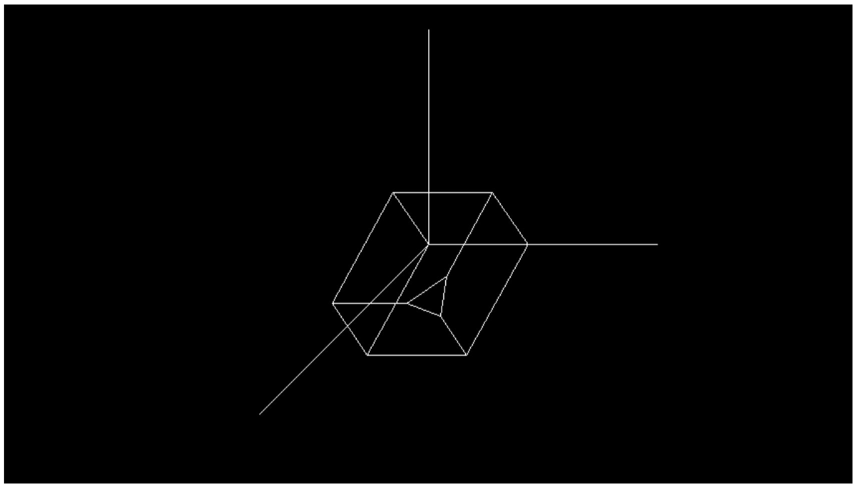

NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

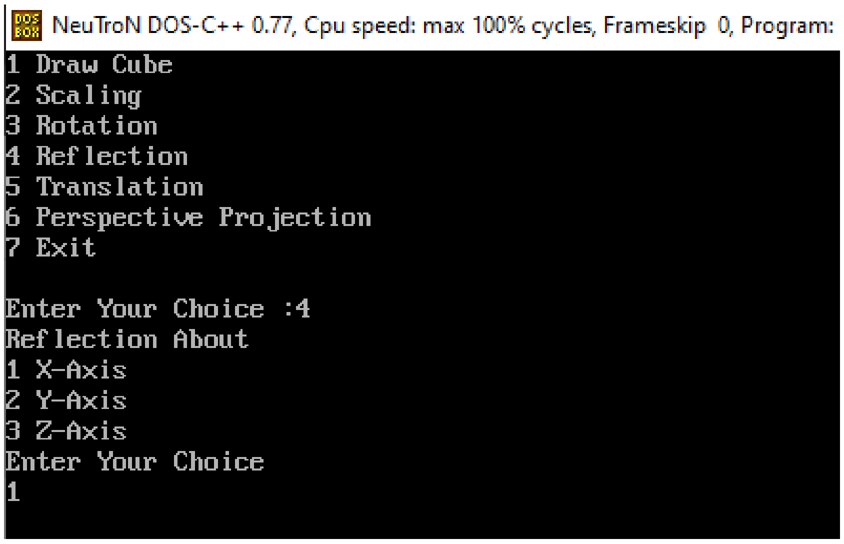
```
1 Draw Cube
2 Scaling
3 Rotation
4 Reflection
5 Translation
6 Perspective Projection
7 Exit

Enter Your Choice :2
Enter The Scaling Factors
2
2
2
```


NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

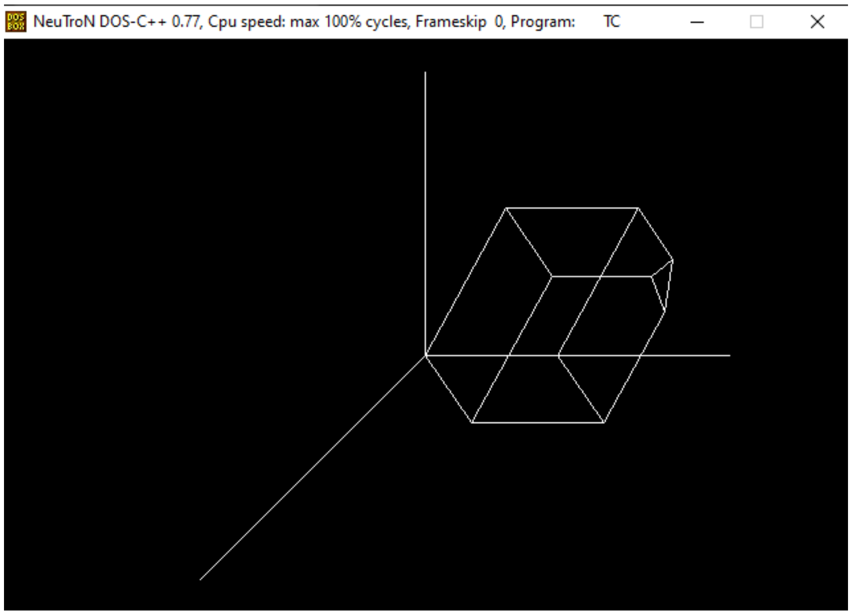


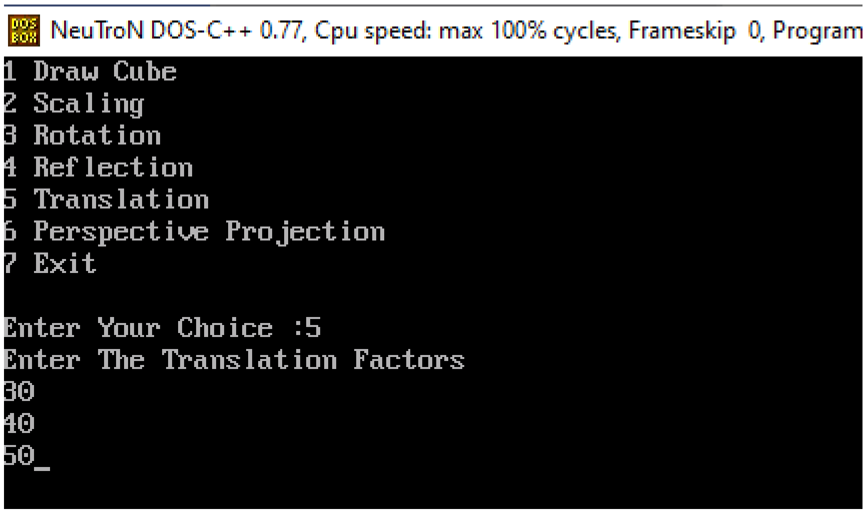
```
NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip  0, Program:     TC
1 Draw Cube
2 Scaling
3 Rotation
4 Reflection
5 Translation
6 Perspective Projection
7 Exit

Enter Your Choice :3
Rotation About
1 X-Axis
2 Y-Axis
3 Z-Axis
Enter Your Choice
1
 Enter The Angle 30
```



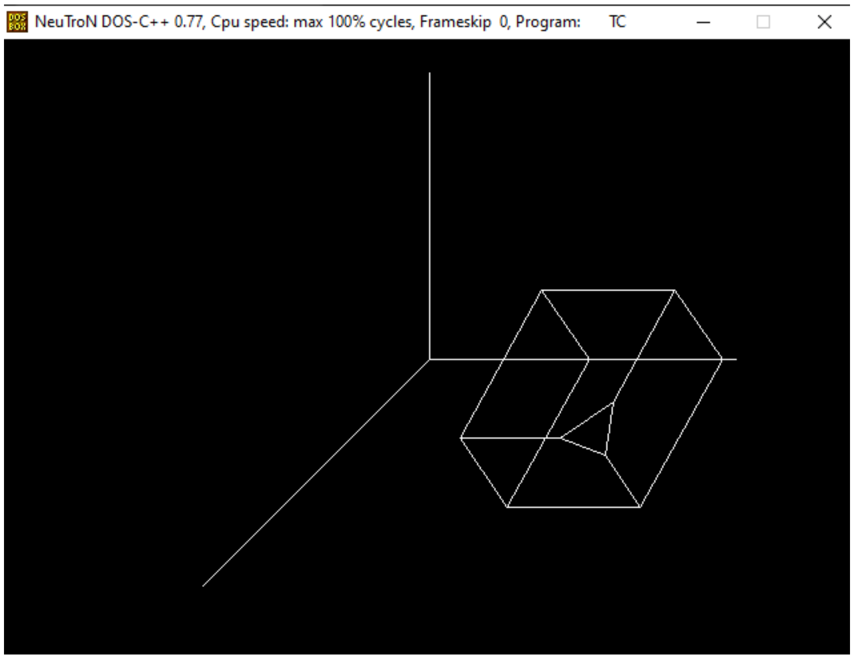


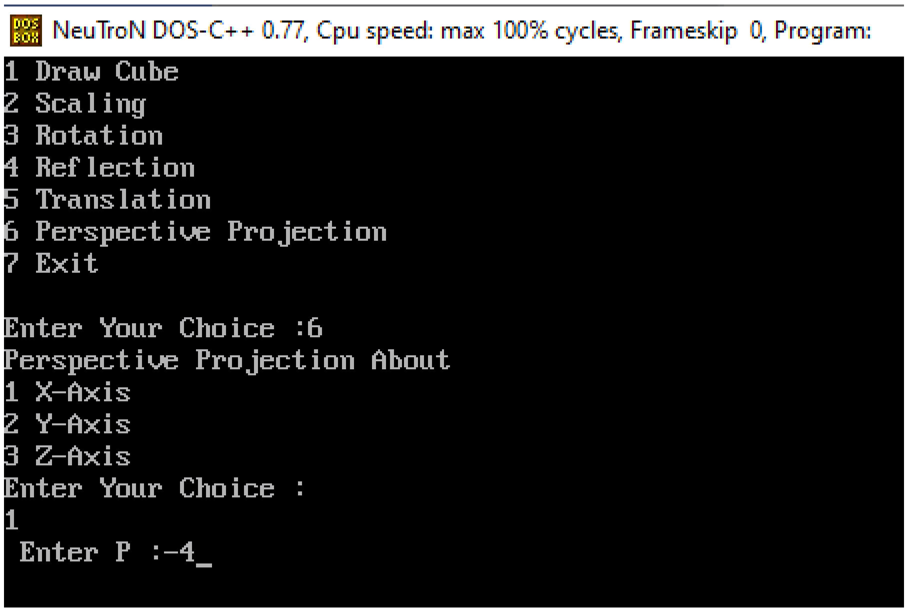
```
NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip  0, Program:
1 Draw Cube
2 Scaling
3 Rotation
4 Reflection
5 Translation
6 Perspective Projection
7 Exit

Enter Your Choice :4
Reflection About
1 X-Axis
2 Y-Axis
3 Z-Axis
Enter Your Choice
1
```


NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

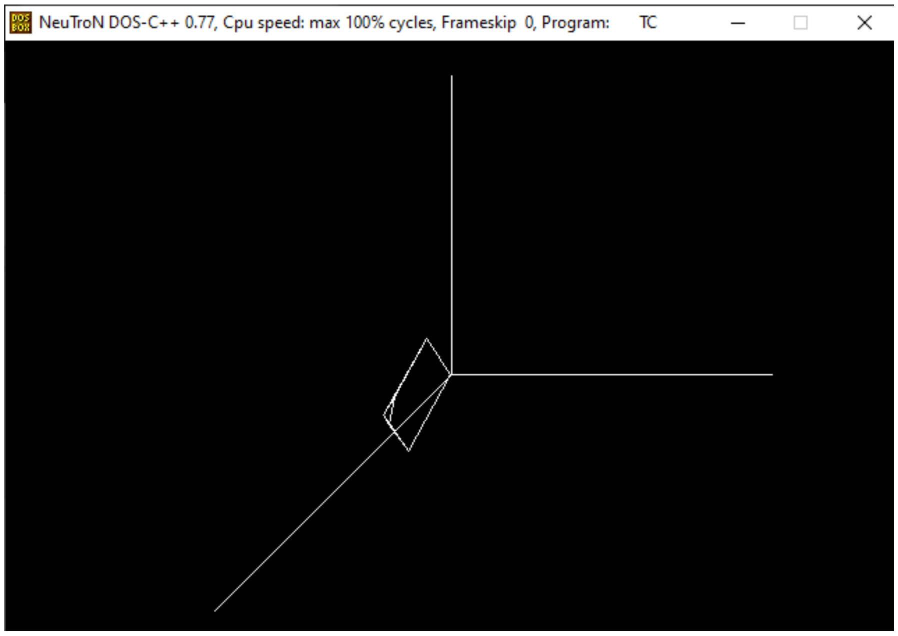

NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program

```
1 Draw Cube
2 Scaling
3 Rotation
4 Reflection
5 Translation
6 Perspective Projection
7 Exit

Enter Your Choice :5
Enter The Translation Factors
30
40
50_
```


NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC


NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program:

```
1 Draw Cube
2 Scaling
3 Rotation
4 Reflection
5 Translation
6 Perspective Projection
7 Exit

Enter Your Choice :6
Perspective Projection About
1 X-Axis
2 Y-Axis
3 Z-Axis
Enter Your Choice :
1
 Enter P :-4_
```


NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

Question 8 : Write a program to draw Hermite /Bezier curve.

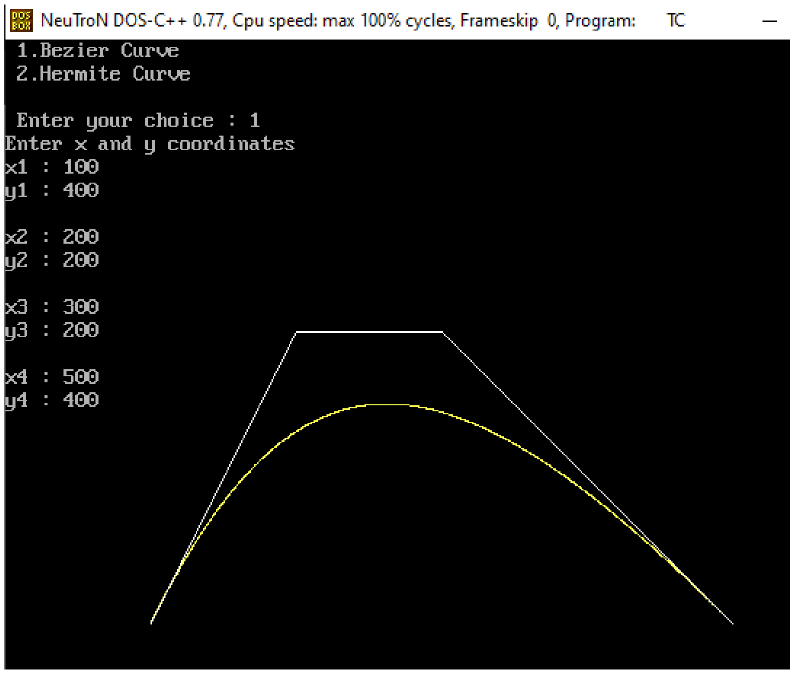
Solution :

Code -

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

//creating Bezier curve function
void bezier_curve(int x[4], int y[4])
{
double t;

for(t=0.0;t<1.0;t=t+0.0005)
{
//Curve Equation of x and y coordinates by using blending function
double xt=pow(1-t,3)*x[0]+3*t*pow(1-
t,2)*x[1]+3*pow(t,2)*(1-t)*x[2]+pow(t,3)*x[3];
double yt=pow(1-t,3)*y[0]+3*t*pow(1-
t,2)*y[1]+3*pow(t,2)*(1-t)*y[2]+pow(t,3)*y[3];
putpixel(xt,yt,YELLOW);
}

for(int i=0;i<3;i++)
{
line(x[i],y[i],x[i+1],y[i+1]);
}
}

//creating Hermite curve function
void hermite_curve(int x1,int y1,int x2,int y2,double t1,double t4)
{
float x,y,t;
for(t=0.0;t<=1.0;t+=0.001)
{
//x and y equation
x=(2*t*t*t-3*t*t+1)*x1+(-2*t*t*t+3*t*t)*x2+(t*t*t- 2*t*t+t)*t1+(t*t*t-t*t)*t4;
y=(2*t*t*t-3*t*t+1)*y1+(-2*t*t*t+3*t*t)*y2+(t*t*t- 2*t*t+1)*t1+(t*t*t-t*t)*t4;
putpixel(x,y,YELLOW);
}
putpixel(x1,y1,GREEN);
putpixel(x2,y2,GREEN);
line(x1,y1,x2,y2);
}

//main function
int main()
{
int gd = DETECT , gm;
initgraph(&gd, &gm,"C:\\TURBOC3\\BGI");
int x1 , y1 , x2 , y2 , n;
```

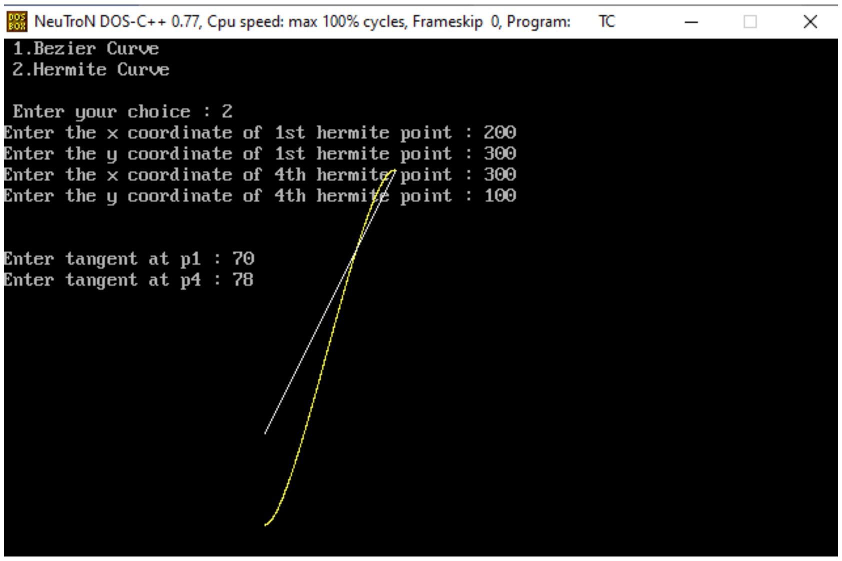
```
double t1,t4;

int x[4],y[4],i;

cout<<" 1.Bezier Curve \n 2.Hermite Curve\n"; cout<<"\n Enter your choice : ";
cin>>n;
if(n==1)
{
//input coordinates of x and y for Bezier curve
cout<<"Enter x and y coordinates \n";
for(i=0;i<4;i++)
{
cout<<"x"<<i+1<<" : ";
cin>>x[i];
cout<<"y"<<i+1<<" : ";
cin>>y[i];
cout<<endl;
}
//calling Bezier curve function
bezier_curve(x,y);
}

else if(n==2)
{ //input coordinates for hermite curve
cout<<"Enter the x coordinate of 1st hermite point : "; cin>>x1;
cout<<"Enter the y coordinate of 1st hermite point : "; cin>>y1;
cout<<"Enter the x coordinate of 4th hermite point : "; cin>>x2;
cout<<"Enter the y coordinate of 4th hermite point : "; cin>>y2;
cout<<"Enter tangent at p1 : ";
cin>>t1;
cout<<"Enter tangent at p4 : ";
cin>>t4;
//calling hermite curve function
hermite_curve(x1,y1,x2,y2,t1,t4);
}
else
{
cout<<"\n Invalid Choice";
}

getch();
return 0;
}
```

Output :



NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
 1.Bezier Curve
 2.Hermite Curve

 Enter your choice : 1
Enter x and y coordinates
x1 : 100
y1 : 400

x2 : 200
y2 : 200

x3 : 300
y3 : 200

x4 : 500
y4 : 400
```



NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
 1.Bezier Curve
 2.Hermite Curve

 Enter your choice : 2
Enter the x coordinate of 1st hermite point : 200
Enter the y coordinate of 1st hermite point : 300
Enter the x coordinate of 4th hermite point : 300
Enter the y coordinate of 4th hermite point : 100


Enter tangent at p1 : 70
Enter tangent at p4 : 78
```

—--------- THE END —--------------