RAMANUJAN COLLEGE

(UNIVERSITY OF DELHI)

COMPUTER GRAPHICS

PRACTICAL FILE

Name – Akansha Jaiswal

Examination Roll No. – 20020570003

Sem – 6th , 3rd year

Class Roll No. – 20201402

TABLE OF CONTENT

S. No.	Questions
1.	write a program to implement Bresenhams line drawing algorithm.
2.	Write a Program to implement mid-point circle drawing algorithm.
3.	Write a program to clip a line using cohen and Sutherland line clipping algorithm.
4.	Write a program to clip a polygon using sutherkand Hodgeman algorithm
5.	Write a program to fill a polygon using Scan line algorithm.
6.	Write a program to apply various 2D transformation on a 2D object.
7.	Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projections on it.
8.	Write a program to draw hermite/Bezier curve.

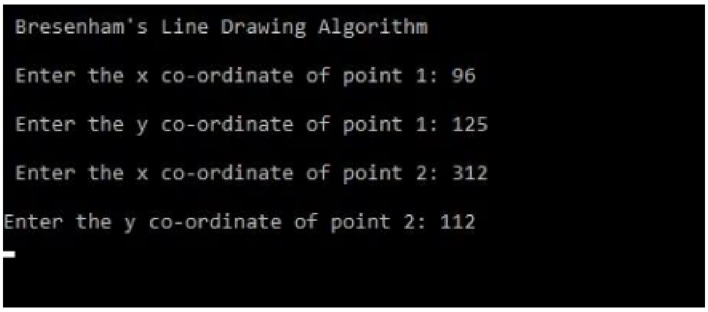
Ques 1 – write a program to implement Bresenhams line drawing algorithm.

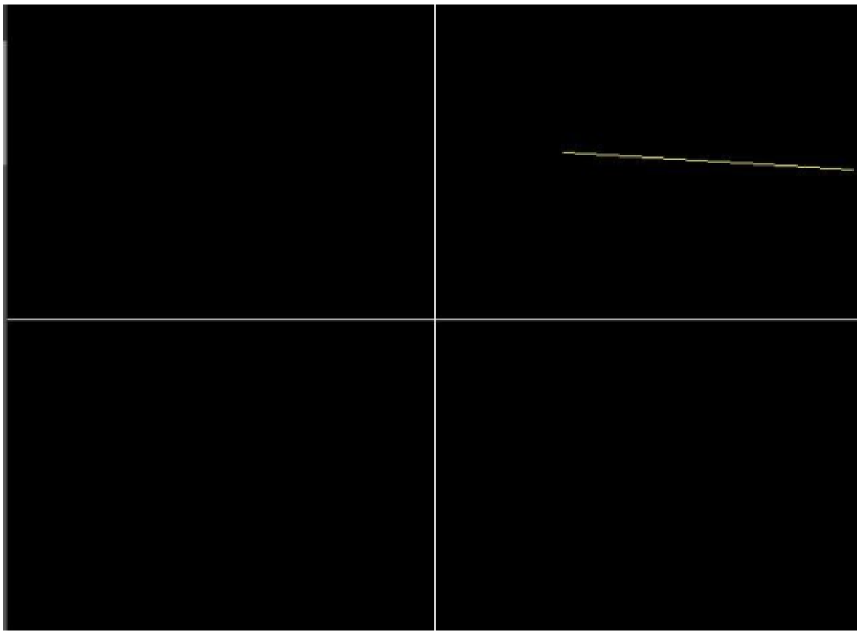
Code:

```
#include<iostream>

#include<bits/stdc++.h>

#include<graphics.h>

using namespace std;


//Function to implement Bresenham's line drawing algorithm

void bresline(int x1,int y1,int x2,int y2)

{

int dx,dy,P,x,y;

int xmid=getmaxx()/2;

int ymid=getmaxy()/2;

dx=x2-x1; dy=y2-

y1;

x=x1;

y=y1;

P=2*dy-dx;

while(x<=x2)

{

 if(P>=0)

 {

 putpixel(x,y,YELLOW);

y=y+1;

 P=P+2*dy-2*dx;

} else {

putpixel(x,y,YELLOW);

P=P+2*dy;} x=x+1;

 }

}
```

```
int main()
{
int gdriver = DETECT,gmode;
initgraph(&gdriver,&gmode,"C:\\Dev-Cpp\\lib");
setbkcolor(BLACK); cleardevice();
int x1,x2,y1,y2;
cout<<" Bresenham's Line Drawing Algorithm \n\n";
cout<<" Enter the x co-ordinate of point 1: ";
cin>>x1;
cout<<"\n Enter the y co-ordinate of point 1: ";
cin>>y1;
cout<<"\n Enter the x co-ordinate of point 2: ";
cin>>x2;
cout<<"\nEnter the y co-ordinate of point 2: ";
cin>>y2;
cleardevice();
 int xmid = getmaxx()/2;
int ymid = getmaxy()/2; line(xmid , 0 , xmid , getmaxy());
line(0 , ymid , getmaxx() , ymid);
bresline(x1+xmid,ymid-y1,x2+xmid,ymid-y2);
getch();
closegraph();
return 0;
}
```

Output :





Ques 2 – Write a Program to implement mid-point circle drawing algorithm.

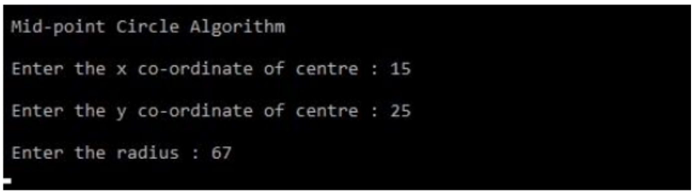
Code :

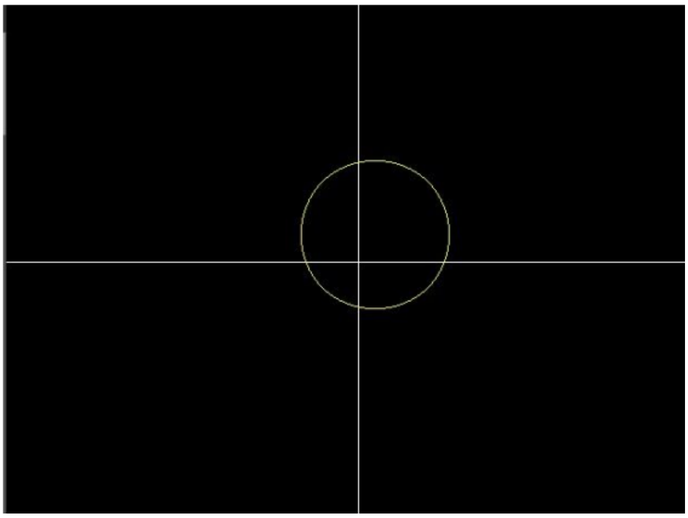
```
#include<iostream>

#include<graphics.h>

#include<math.h>

using namespace std;

void circlePlotPoints (int, int, int, int); int

xmid, ymid;

void circleMidpoint(int xCenter, int yCenter, int radius)

{

int x = 0;

int y = radius; int p

= 1 - radius;
```

```
//circlePlotPoints (x, y, xCenter, yCenter);
while (x <= y)
{
 circlePlotPoints (x, y, xCenter, yCenter);
if (p < 0)
 {
 p += (2*x)+1;
}
else
 { p
+=(2*(x-y))+1; y--
;
 }
x++ ;
}
 }

void circlePlotPoints(int x, int y, int xCenter, int yCenter){
putpixel (xCenter + x, yCenter + y, YELLOW);
putpixel (xCenter - x, yCenter + y, YELLOW); putpixel
(xCenter + x, yCenter - y, YELLOW); putpixel (xCenter - x,
yCenter - y, YELLOW); putpixel (xCenter + y, yCenter + x,
YELLOW); putpixel (xCenter - y, yCenter + x, YELLOW);
putpixel (xCenter + y, yCenter - x, YELLOW); putpixel
(xCenter - y, yCenter - x, YELLOW);
}
int main()
{
int x , y;
 float r;
 int gd = DETECT , gm;
```

```
initgraph(&gd, &gm, (char*)"");

cout<<" Mid-point Circle Algorithm \n\n";

cout<<" Enter the x co-ordinate of centre : ";

cin>>x;

cout<<"\n Enter the y co-ordinate of centre : ";

cin>>y;

cout<<"\n Enter the radius : ";

cin>>r;

 xmid = getmaxx()/2;

ymid = getmaxy()/2; line(xmid ,

0 , xmid , getmaxy()); line(0

, ymid , getmaxx() , ymid);

circleMidpoint(x + xmid , ymid -

y , r);

 getch();

closegraph(); return

0;

}
```

Output :





Ques 3 – Write a program to clip a line using cohen and Sutherland line clipping algorithm.

<u>Code :</u>

```
// C++ program to implement Cohen Sutherland algorithm
// for line clipping.
// including libraries
#include <bits/stdc++.h>
#include <graphics.h>
using namespace std;

// Global Variables
int xmin, xmax, ymin, ymax;

// Lines where co-ordinates are (x1, y1) and (x2, y2)
struct lines {
   int x1, y1, x2, y2;
};

// This will return the sign required.
int sign(int x)
{
   if (x > 0)
      return 1;
   else
      return 0;
}

// CohenSutherLand LineClipping Algorithm As Described in theory.
// This will clip the lines as per window boundaries.
void clip(struct lines mylines)
```

```
{
    // arrays will store bits
    // Here bits implies initial Point whereas bite implies end points
    int bits[4], bite[4], i, var;
    // setting color of graphics to be RED
    setcolor(RED);

    // Finding Bits
    bits[0] = sign(xmin - mylines.x1);
    bite[0] = sign(xmin - mylines.x2);
    bits[1] = sign(mylines.x1 - xmax);
    bite[1] = sign(mylines.x2 - xmax);
    bits[2] = sign(ymin - mylines.y1);
    bite[2] = sign(ymin - mylines.y2);
    bits[3] = sign(mylines.y1 - ymax);
    bite[3] = sign(mylines.y2 - ymax);

    // initial will used for initial coordinates and end for final
    string initial = "", end = "", temp = "";

    // convert bits to string
    for (i = 0; i < 4; i++) {
        if (bits[i] == 0)
            initial += '0';
        else
            initial += '1';
    }
    for (i = 0; i < 4; i++) {
        if (bite[i] == 0)
            end += '0';
        else
```

```
            end += '1';
    }

    // finding slope of line y=mx+c as (y-y1)=m(x-x1)+c
    // where m is slope m=dy/dx;

    float m = (mylines.y2 - mylines.y1) / (float)(mylines.x2 - mylines.x1);
    float c = mylines.y1 - m * mylines.x1;

    // if both points are inside the Accept the line and draw
    if (initial == end && end == "0000") {
        // inbuild function to draw the line from(x1, y1) to (x2, y2)
        line(mylines.x1, mylines.y1, mylines.x2, mylines.y2);
        return;
    }

    // this will contain cases where line maybe totally outside for partially inside
    else {
        // taking bitwise end of every value
        for (i = 0; i < 4; i++) {

            int val = (bits[i] & bite[i]);
            if (val == 0)
                temp += '0';
            else
                temp += '1';
        }
        // as per algo if AND is not 0000 means line is completely outside hence draw nothing and
return
        if (temp != "0000")
            return;
```

```
// Here contain cases of partial inside or outside
// So check for every boundary one by one
for (i = 0; i < 4; i++) {
   // if boths bit are same hence we cannot find any intersection with boundary so continue
   if (bits[i] == bite[i])
      continue;
   // Otherwise there exist a intersection

   // Case when initial point is in left xmin
   if (i == 0 && bits[i] == 1) {
      var = round(m * xmin + c);
      mylines.y1 = var;
      mylines.x1 = xmin;
   }
   // Case when final point is in left xmin
   if (i == 0 && bite[i] == 1) {
      var = round(m * xmin + c);
      mylines.y2 = var;
      mylines.x2 = xmin;
   }
   // Case when initial point is in right of xmax
   if (i == 1 && bits[i] == 1) {
      var = round(m * xmax + c);
      mylines.y1 = var;
      mylines.x1 = xmax;
   }
   // Case when final point is in right of xmax
   if (i == 1 && bite[i] == 1) {
      var = round(m * xmax + c);
      mylines.y2 = var;
```

```
    mylines.x2 = xmax;
}
// Case when initial point is in top of ymin
if (i == 2 && bits[i] == 1) {
    var = round((float)(ymin - c) / m);
    mylines.y1 = ymin;
    mylines.x1 = var;
}
// Case when final point is in top of ymin
if (i == 2 && bite[i] == 1) {
    var = round((float)(ymin - c) / m);
    mylines.y2 = ymin;
    mylines.x2 = var;
}
// Case when initial point is in bottom of ymax
if (i == 3 && bits[i] == 1) {
    var = round((float)(ymax - c) / m);
    mylines.y1 = ymax;
    mylines.x1 = var;
}
// Case when final point is in bottom of ymax
if (i == 3 && bite[i] == 1) {
    var = round((float)(ymax - c) / m);
    mylines.y2 = ymax;
    mylines.x2 = var;
}
// Updating Bits at every point
bits[0] = sign(xmin - mylines.x1);
bite[0] = sign(xmin - mylines.x2);
bits[1] = sign(mylines.x1 - xmax);
bite[1] = sign(mylines.x2 - xmax);
```

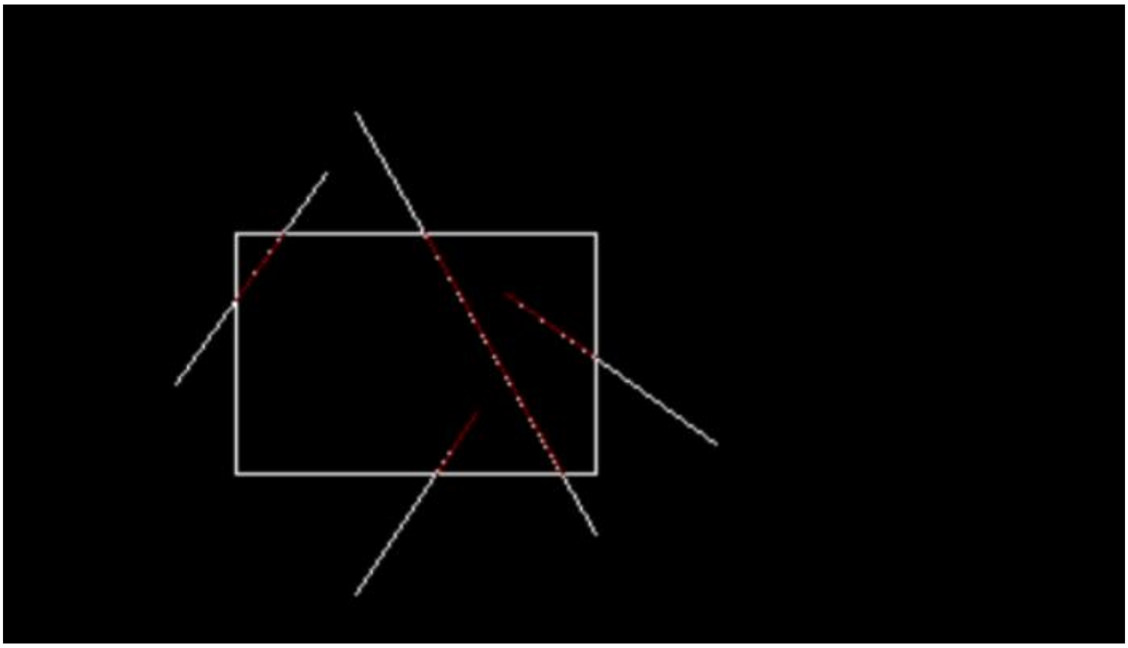
```
        bits[2] = sign(ymin - mylines.y1);
        bite[2] = sign(ymin - mylines.y2);
        bits[3] = sign(mylines.y1 - ymax);
        bite[3] = sign(mylines.y2 - ymax);
    } // end of for loop
    // Initialize initial and end to NULL
    initial = "", end = "";
    // Updating strings again by bit
    for (i = 0; i < 4; i++) {
        if (bits[i] == 0)
            initial += '0';
        else
            initial += '1';
    }
    for (i = 0; i < 4; i++) {
        if (bite[i] == 0)
            end += '0';
        else
            end += '1';
    }
    // If now both points lie inside or on boundary then simply draw the updated line
    if (initial == end && end == "0000") {
        line(mylines.x1, mylines.y1, mylines.x2, mylines.y2);
        return;
    }
    // else line was completely outside hence rejected
    else
        return;
  }
}
```

```
// Driver Function
int main()
{
   int gd = DETECT, gm;

   // Setting values of Clipping window
   xmin = 80;
   xmax = 200;
   ymin = 80;
   ymax = 160;

   // Setup
   int win = initwindow(400, 300, "Line Clipping Example");
   setcurrentwindow(win);



   // Drawing Window using Lines
   line(xmin, ymin, xmax, ymin);
   line(xmax, ymin, xmax, ymax);
   line(xmax, ymax, xmin, ymax);
   line(xmin, ymax, xmin, ymin);

   // Assume 4 lines to be clipped
   struct lines mylines[4];

   // Setting the coordinated of  4 lines
   mylines[0].x1 = 60;
   mylines[0].y1 = 130;
   mylines[0].x2 = 110;
   mylines[0].y2 = 60;
```

```
mylines[1].x1 = 120;
mylines[1].y1 = 40;
mylines[1].x2 = 200;
mylines[1].y2 = 180;

mylines[2].x1 = 120;
mylines[2].y1 = 200;
mylines[2].x2 = 160;
mylines[2].y2 = 140;

mylines[3].x1 = 170;
mylines[3].y1 = 100;
mylines[3].x2 = 240;
mylines[3].y2 = 150;

// Drawing Initial Lines without clipping
for (int i = 0; i < 4; i++) {
   line(mylines[i].x1, mylines[i].y1,
        mylines[i].x2, mylines[i].y2);
   delay(1000);
}

// Drawing clipped Line
for (int i = 0; i < 4; i++) {
   // Calling clip() which in term clip the line as per window and draw it
   clip(mylines[i]);
   delay(1000);
}
delay(4000);
getch();
// For Closing the graph.
```

```
   closegraph();
   return 0;
}
```

Output :



Ques 4 – Write a program to clip a polygon using sutherkand Hodgeman algorithm .

Code :

```
#include<iostream>
#include <graphics.h>
using namespace std;

const int MAX_POINTS = 20;

// Returns x-value of point of intersection of two
// lines
int x_intersect(int x1, int y1, int x2, int y2,
            int x3, int y3, int x4, int y4)
{
   int num = (x1*y2 - y1*x2) * (x3-x4) -
```

```
          (x1-x2) * (x3*y4 - y3*x4);
    int den = (x1-x2) * (y3-y4) - (y1-y2) * (x3-x4);
    return num/den;
}

// Returns y-value of point of intersection of
// two lines
int y_intersect(int x1, int y1, int x2, int y2,
                int x3, int y3, int x4, int y4)
{
    int num = (x1*y2 - y1*x2) * (y3-y4) -
          (y1-y2) * (x3*y4 - y3*x4);
    int den = (x1-x2) * (y3-y4) - (y1-y2) * (x3-x4);
    return num/den;
}

// This functions clips all the edges w.r.t one clip
// edge of clipping area
void clip(int poly_points[][2], int &poly_size,
        int x1, int y1, int x2, int y2)
{
    int new_points[MAX_POINTS][2], new_poly_size = 0;

    // (ix,iy),(kx,ky) are the co-ordinate values of
    // the points
    for (int i = 0; i < poly_size; i++)
    {
        // i and k form a line in polygon
        int k = (i+1) % poly_size;
        int ix = poly_points[i][0], iy = poly_points[i][1];
        int kx = poly_points[k][0], ky = poly_points[k][1];
```

```
// Calculating position of first point
// w.r.t. clipper line
int i_pos = (x2-x1) * (iy-y1) - (y2-y1) * (ix-x1);

// Calculating position of second point
// w.r.t. clipper line
int k_pos = (x2-x1) * (ky-y1) - (y2-y1) * (kx-x1);

// Case 1 : When both points are inside
if (i_pos < 0  && k_pos < 0)
{
   //Only second point is added
   new_points[new_poly_size][0] = kx;
   new_points[new_poly_size][1] = ky;
   new_poly_size++;
}

// Case 2: When only first point is outside
else if (i_pos >= 0  && k_pos < 0)
{
   // Point of intersection with edge
   // and the second point is added
   new_points[new_poly_size][0] = x_intersect(x1,
                 y1, x2, y2, ix, iy, kx, ky);
   new_points[new_poly_size][1] = y_intersect(x1,
                 y1, x2, y2, ix, iy, kx, ky);
   new_poly_size++;

   new_points[new_poly_size][0] = kx;
   new_points[new_poly_size][1] = ky;
```

```
            new_poly_size++;
        }

        // Case 3: When only second point is outside
        else if (i_pos < 0  && k_pos >= 0)
        {
            //Only point of intersection with edge is added
            new_points[new_poly_size][0] = x_intersect(x1,
                            y1, x2, y2, ix, iy, kx, ky);
            new_points[new_poly_size][1] = y_intersect(x1,
                            y1, x2, y2, ix, iy, kx, ky);
            new_poly_size++;
        }

        // Case 4: When both points are outside
        else
        {
            //No points are added
        }
    }

    // Copying new points into original array
    // and changing the no. of vertices
    poly_size = new_poly_size;
    for (int i = 0; i < poly_size; i++)
    {
        poly_points[i][0] = new_points[i][0];
        poly_points[i][1] = new_points[i][1];
    }
}
```

```
// Implements Sutherland–Hodgman algorithm
void suthHodgClip(int poly_points[][2], int poly_size,
              int clipper_points[][2], int clipper_size)
{
   //i and k are two consecutive indexes
   for (int i=0; i<clipper_size; i++)
   {
      int k = (i+1) % clipper_size;

      // We pass the current array of vertices, it's size
      // and the end points of the selected clipper line
      clip(poly_points, poly_size, clipper_points[i][0],
          clipper_points[i][1], clipper_points[k][0],
          clipper_points[k][1]);
   }

   // Printing vertices of clipped polygon
   cout << "\nClipped Polygon : " << endl;
   for (int i=0; i < poly_size; i++)
      cout << '(' << poly_points[i][0] <<
            ", " << poly_points[i][1] << ") ";
   cout << endl << endl;

   // Drawing Clipped Polygon
   int poly_clipped[50];
   for (int q = 0; q < poly_size; q++)
   {
      for (int t = 0; t < 2; t++)
      {
         poly_clipped[q * 2 + t] = poly_points[q][t];
      }
```

```
    }
    setcolor(BLUE);
    poly_clipped[2 * poly_size] = poly_clipped[0];
    poly_clipped[2 * poly_size + 1] = poly_clipped[1];
    drawpoly(poly_size + 1, poly_clipped);

    getch();
}

//Driver code
int main()
{
    int gd = DETECT, gm, errorcode;
    initgraph(&gd, &gm, NULL);

    // Defining polygon vertices in clockwise order
    int poly_size = 3;
    int poly_points[20][2] = {{100,150}, {200,250},
                              {300,100}};

    // Defining clipper polygon vertices in clockwise order
    // 1st Example with square clipper
    int clipper_size = 4;
    int clipper_points[][2] = {{100,100}, {100,200},
                              {200,200}, {200,100} };
    setcolor(RED);
    rectangle(100, 100, 200, 200);

    setcolor(YELLOW);
    int poly[50];
    for (int q = 0; q < poly_size; q++)
```

```
  {
    for (int t = 0; t < 2; t++)
    {
      poly[q * 2 + t] = poly_points[q][t];
    }
  }
  poly[2 * poly_size] = poly[0];
  poly[2 * poly_size + 1] = poly[1];
  drawpoly(poly_size + 1, poly);

  //Calling the clipping function
  suthHodgClip(poly_points, poly_size, clipper_points,
          clipper_size);

  getch();

  return 0;
}
```

Output :

Ques 5 – Write a program to fill a polygon using Scan line algorithm.

Code :

```
#include<iostream>

#include<graphics.h>

#include<math.h> using

namespace std;

const int WINDOW_HEIGHT = 1000;

typedef struct tdcPt

{

int x;

int y;

}dcPt;

typedef struct tEdge

{

int yUpper;

float xIntersect, dxPerScan;

struct tEdge *next;

}Edge;


// Vertices: Array of structures.

dcPt vertex[5] = {{200, 500}, {300, 250}, {270, 230}, {320, 200}, {360, 290}};

void insertEdge(Edge *list, Edge *edge)

{

Edge *p, *q = list; p = q-

>next;

while (p != NULL)

{

if (edge->xIntersect < p->xIntersect)

p = NULL; else

{

q = p;
```

```
p = p->next;
}
}
edge->next = q->next; q->next =
edge;
}
int yNext(int k, int cnt, dcPt *pts)
{
int j;
if ((k + 1) > (cnt - 1))
j = 0;
else j = k +
1; while(pts[k].y ==
pts[j].y)
{
if ((j + 1) > (cnt - 1))
j =
0; else
j++;
}
return (pts[j].y);
}
void makeEdgeRec(dcPt lower, dcPt upper, int yComp, Edge *edge, Edge
*edges[])
{
edge->dxPerScan = (float) (upper.x - lower.x) / (upper.y -
lower.y); edge->xIntersect = lower.x; if (upper.y < yComp)
edge->yUpper = upper.y - 1; else
edge->yUpper = upper.y;
insertEdge(edges[lower.y], edge);
}
```

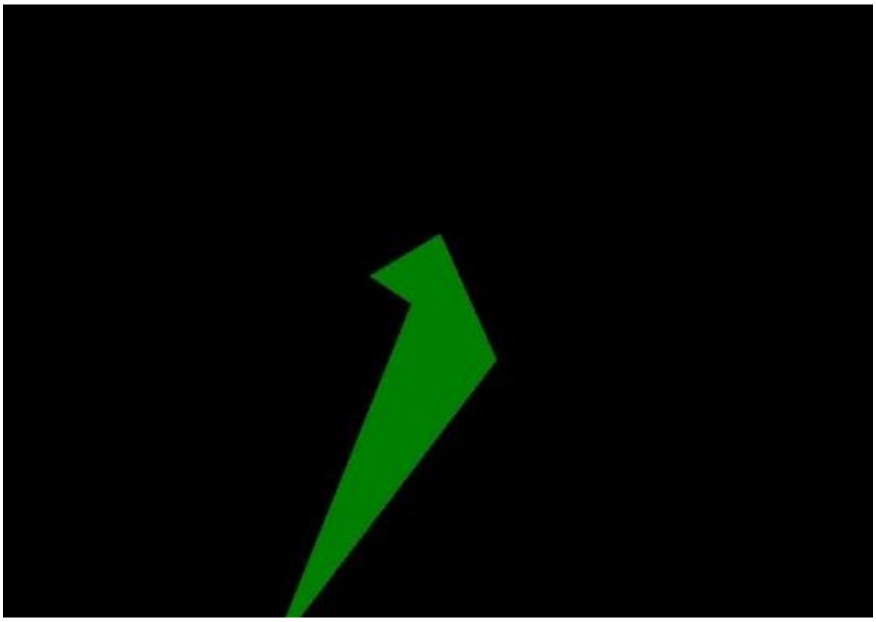
```
void buildEdgeList(int cnt, dcPt *pts, Edge *edges[])
{
Edge *edge;
dcPt v1, v2; int i, yPrev
= pts[cnt - 2].y;
v1.x = pts[cnt - 1].x; v1.y = pts[cnt - 1].y;
for(int i = 0; i < cnt; i++)
{
v2 = pts[i];
 if (v1.y != v2.y) // nonhorizontal line
{
edge = (Edge *) malloc (sizeof(Edge));
if (v1.y < v2.y) //
upgoing edge
makeEdgeRec(v1, v2, yNext(i, cnt, pts), edge, edges);
else // down�going edge
makeEdgeRec(v2, v1 , yPrev, edge, edges);
}
yPrev = v1.y;
v1 = v2;
}
}
void buildActiveList(int scan, Edge *active, Edge *edges[])
{
Edge *p, *q;
p = edges[scan]->next;
while (p)
{
q = p->next; insertEdge(active, p);
p = q;
}
```

```
}
void fillScan(int scan, Edge *active)
{
Edge *p1, *p2 ;
int i;
p1 = active->next; while (p1)
{
p2 = p1->next;
 for(i = p1->xIntersect; i < p2->xIntersect; i++)
putpixel((int) i, scan, GREEN); p1 = p2->next;
}
}
void deleteAfter(Edge *q)
{
Edge *p = q->next;
q->next = p->next;
free(p);
}
void updateActiveList(int scan, Edge *active)
{
Edge *q = active, *p = active->next;
while (p)
{
if (scan >= p->yUpper)
{
p = p->next; deleteAfter(q);
}
else
{
p->xIntersect = p->xIntersect + p->dxPerScan;
q = p;
```

```
p = p->next;
}
}
}
void resortActiveList(Edge *active)
{
Edge *q, *p = active->next;
active->next = NULL; while (p)
{
q = p->next;
insertEdge(active, p);
p = q;
}
}
void scanFill(int cnt, dcPt *pts)
{
Edge *edges[WINDOW_HEIGHT], *active;
int i, scan;
for (i = 0; i < WINDOW_HEIGHT; i++)
{
 edges[i] = (Edge *) malloc (sizeof(Edge));; edges[i]->next = NULL;
}
buildEdgeList(cnt, pts, edges); active = (Edge
*) malloc (sizeof(Edge));; active->next =
NULL;
for (scan = 0; scan < WINDOW_HEIGHT; scan++) {
buildActiveList(scan, active, edges); if
(active->next)
{
 fillScan(scan, active); updateActiveList(scan,
active) ; resortActiveList(active);
```

```
}
}
free(edges[WINDOW_HEIGHT]);
free(active);
}
int main()
{
int gd = DETECT, gm;
initgraph(&gd, &gm, (char*)"");
float X = getmaxx(), Y =
getmaxy(); float x_mid = X / 2;
float y_mid = Y / 2;
cleardevice();
scanFill(5, vertex);
getch();
closegraph(); return
0;
}
```

Output :



Ques 6 – Write a program to apply various 2D transformation on a 2D object.

Code –

```
#include<graphics.h>

#include<stdlib.h>

#include<stdio.h>

#include<iostream>

#include<conio.h>

#include<math.h> using

namespace std;

int mat[3][3];

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);

}

}

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;
}
}
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]);
}
}
}
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 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;
}
} 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 (
```

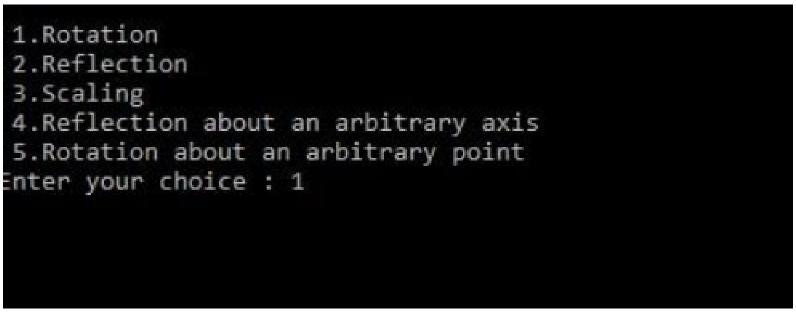
```
j=0;j<2;j++){ for ( k=0;k<2;k++){ c[i][j]=c[i][j]+(a[i][k]*t[k][j]);
}
}
}
dda_line(c[0][0],c[0][1],c[1][0],c[1][1],GREEN);
dda_line(c[1][0],c[1][1],c[2][0],c[2][1],GREEN);
dda_line(c[2][0],c[2][1],c[0][0],c[0][1],GREEN);
}
void scaling(){ int xmid , ymid; xmid
= getmaxx()/2; ymid = getmaxy()/2;
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;
}
}
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]);
}
}
}
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);
}
```

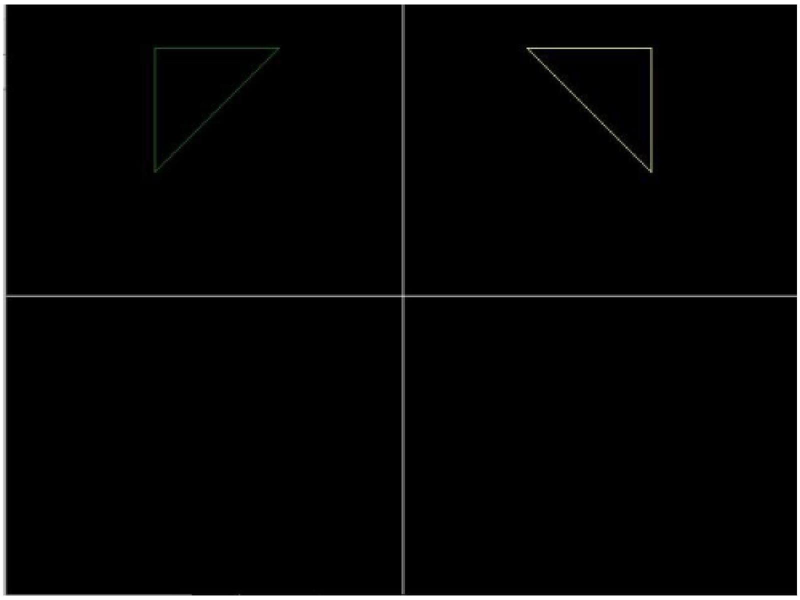
```
void multi(int a[3][3] , int b[3][3] ){
int i , j ,k; int c[3][3];
for( i = 0 ; i < 3 ;
i++){ for(j=0 ; j< 3 ;
j++){ c[i][j]=0;
}
}
for ( i=0;i<3;i++){ for
( j=0;j<3;j++){ for (
k=0;k<3;k++){
c[i][j]=c[i][j]+(a[i][k]
*b[k][j]);
}
}
}
for( i = 0 ; i < 3 ; i++){ for(j=0
; j< 3 ; j++){ mat[i][j]=c[i][j];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

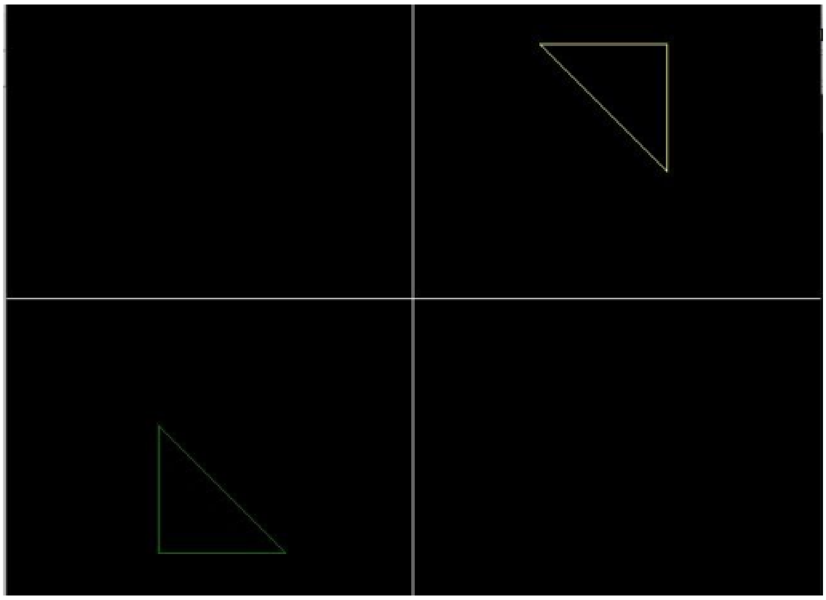
```
c[i][j]=c[i][j]+(a[i][k]*mat[k][j])
;
}
}
}
dda_line(c[0][0],c[0][1],c[1][0],c[1][1],GREEN);
dda_line(c[1][0],c[1][1],c[2][0],c[2][1],GREEN);
dda_line(c[2][0],c[2][1],c[0][0],c[0][1],GREEN);
}
int main()
{
int gdriver = DETECT , gmode , errorcode;
initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI"); 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();
}
```

Output :

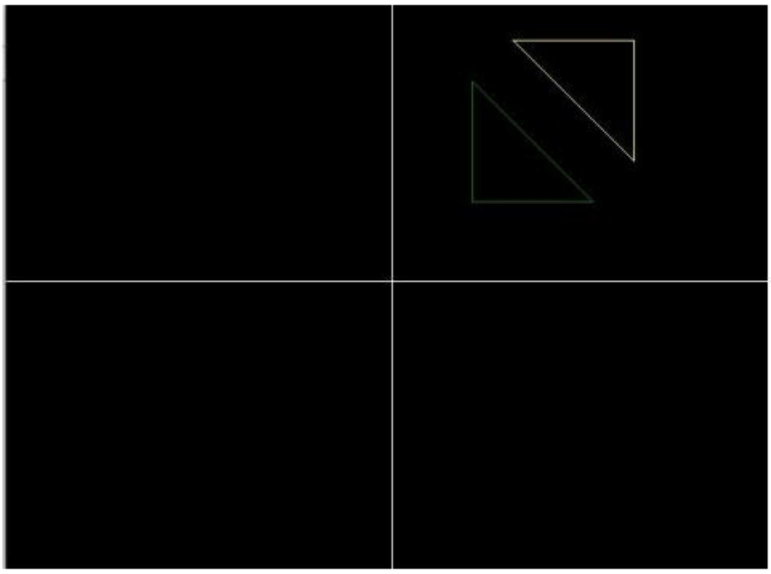

1.Rotation
2.Reflection
3.Scaling
4.Reflection about an arbitrary axis
5.Rotation about an arbitrary point
Enter your choice : 1



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



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



Ques 7 – Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projections on it.

Code :

```
#include<iostream>

#include<dos.h>

#include<stdio.h>

#include<math.h>

#include<conio.h>

#include<graphics.h>

#include<process.h>

double x1,x2,y1,y2; void

draw_cube(double

edge[20][3]){

int i;

cleardevice(); for(i=0;i<19;i++){

x1=edge[i][0]+edge[i][2]*(cos(2.3562));
```

```
y1=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(x1+320,240-
y1,x2+320,240-y2);
}
line(320,240,320,25); line(320,240,550,240);
line(320,240,150,410);
}
void translate(double edge[20][3]){
int a,b,c; int i;
cout<<"Enter the Translation Factors : ";
cin>>a>>b>>c; cleardevice();
for(i=0;i<20;i++){ edge[i][0]+=a;
edge[i][0]+=b; edge[i][0]+=c;
}
draw_cube(edge);
}
void rotate(double edge[20][3]){
int n; int i;
double temp,theta,temp1; cleardevice();
cout<<" 1.X-Axis \n 2.Y-Axis \n 3.Z-Axis \n";
cout<<"Enter your choice : "; cin>>n;
switch(n){
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];
```

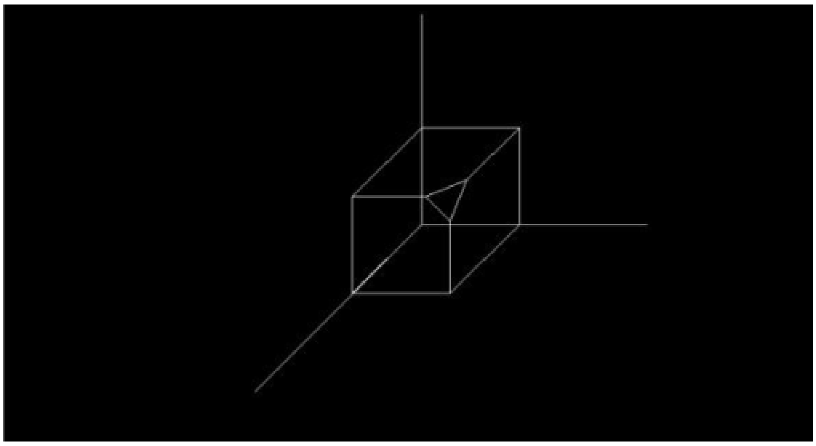
```
edge[i][1]=temp*cos(theta)-temp1*sin(theta);
edge[i][2]=temp*sin(theta)+temp1*cos(theta);
}
draw_cube(edge); break;
case 2: cout<<" Enter The Angle ";
cin>>theta;
theta=(theta*3.14)/180;
for(i=0;i<20;i++){
edge[i][1]=edge[i][1];
temp=edge[i][0]; temp1=edge[i][2];
edge[i][0]=temp*cos(theta)+temp1*sin(theta); edge[i][2]=-
temp*sin(theta)+temp1*cos(theta);
}
draw_cube(edge); break;
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];
edge[i][0]=temp*cos(theta)-temp1*sin(theta);
edge[i][1]=temp*sin(theta)+temp1*cos(theta);
}
draw_cube(edge); break;
}
}
void reflect(double edge[20][3]){
int n; int i;
cleardevice();
cout<<" 1.X-Axis \n 2.Y-Axis \n 3.Z-Axis \n";
```

```
cout<<" Enter Your Choice : "; cin>>n;
switch(n){ 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; 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; 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;
}
}
void perspect(double edge[20][3]){
int n; int i;
double p,q,r; cleardevice();
cout<<" 1.X-Axis \n 2.Y-Axis \n 3.Z-Axis\n";
cout<<" Enter Your Choice : "; cin>>n;
switch(n){ 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);
```

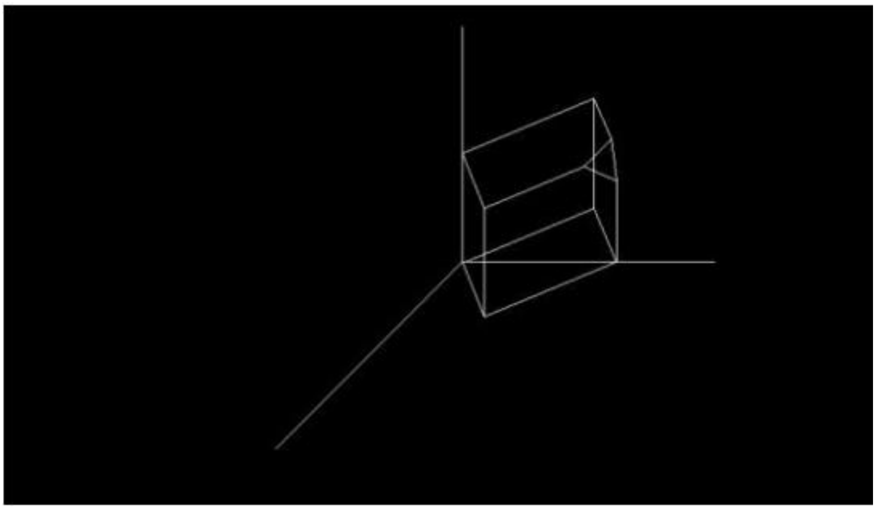
```
edge[i][2]=edge[i][2]/(p*edge[i][0]+1);
}
draw_cube(edge); break; case 2:
cout<<" Enter Q : "; cin>>q;
for(i=0;i<20;i++){
edge[i][1]=edge[i][1]/(edge[i][1]*q+1)
;
edge[i][0]=edge[i][0]/(edge[i][1]*q+1)
;
edge[i][2]=edge[i][2]/(edge[i][1]*q+1)
;
}
draw_cube(edge); break;
case 3: cout<<" Enter R : ";
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;
}
}
void main(){ clrscr();
int gdriver = DETECT , gmode , errorcode;
initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI");
int n;
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};
```

```
cout<<" 1.Draw Cube \n 2.Rotation \n 3.Reflection \n"; cout<<"
4.Translation \n 5.Perspective Projection \n"; cout<<" Enter Your
Choice : ";
cin>>n; switch(n){ case 1:
draw_cube(edge); break; case 2:
rotate(edge); break; case 3:
reflect(edge); break; case 4:
translate(edge); break; case 5:
perspect(edge); break; default:
cout<<" Invalid Choice\n ";
}
getch();
}
```

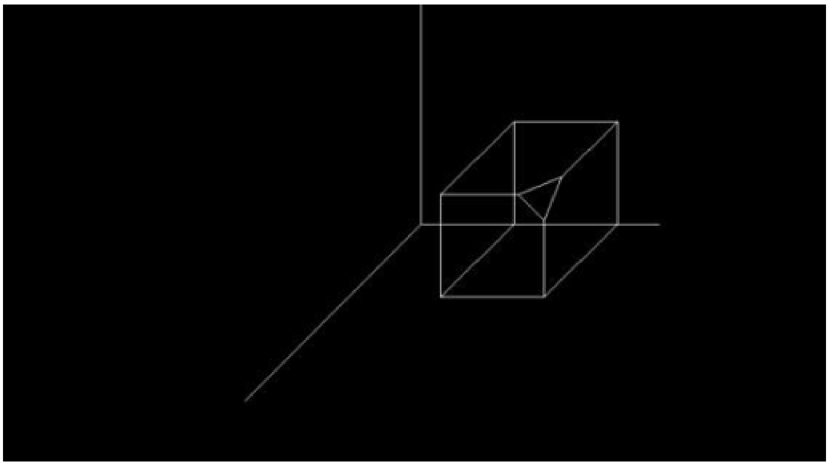
Output :



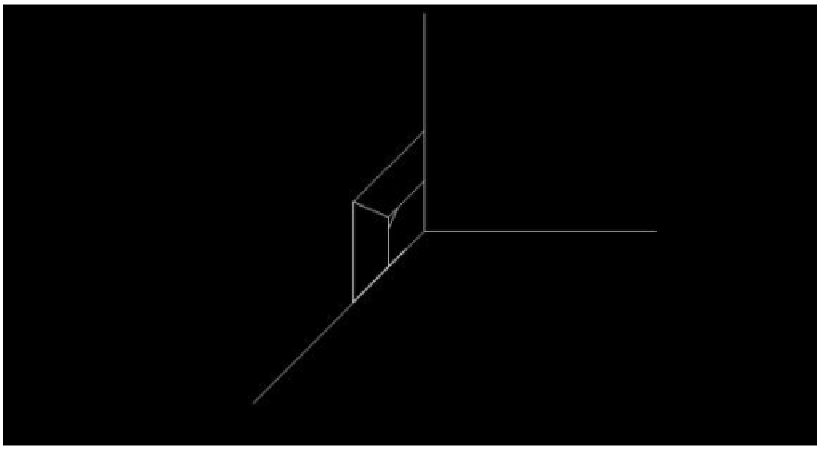
ROTATION ABOUT Y-AXIS BY AN ANGLE OF 45DEGREE



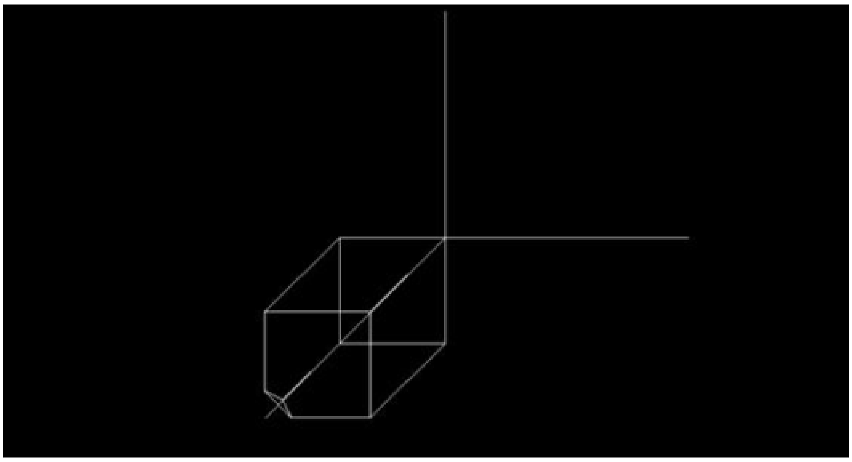
TRANSLATION FACTORS AS 20,30,40



PERSPECTIVE PROJECTION ABOUT X-AXIS WHEN P=50



REFLECTION ABOUT Z-AXIS



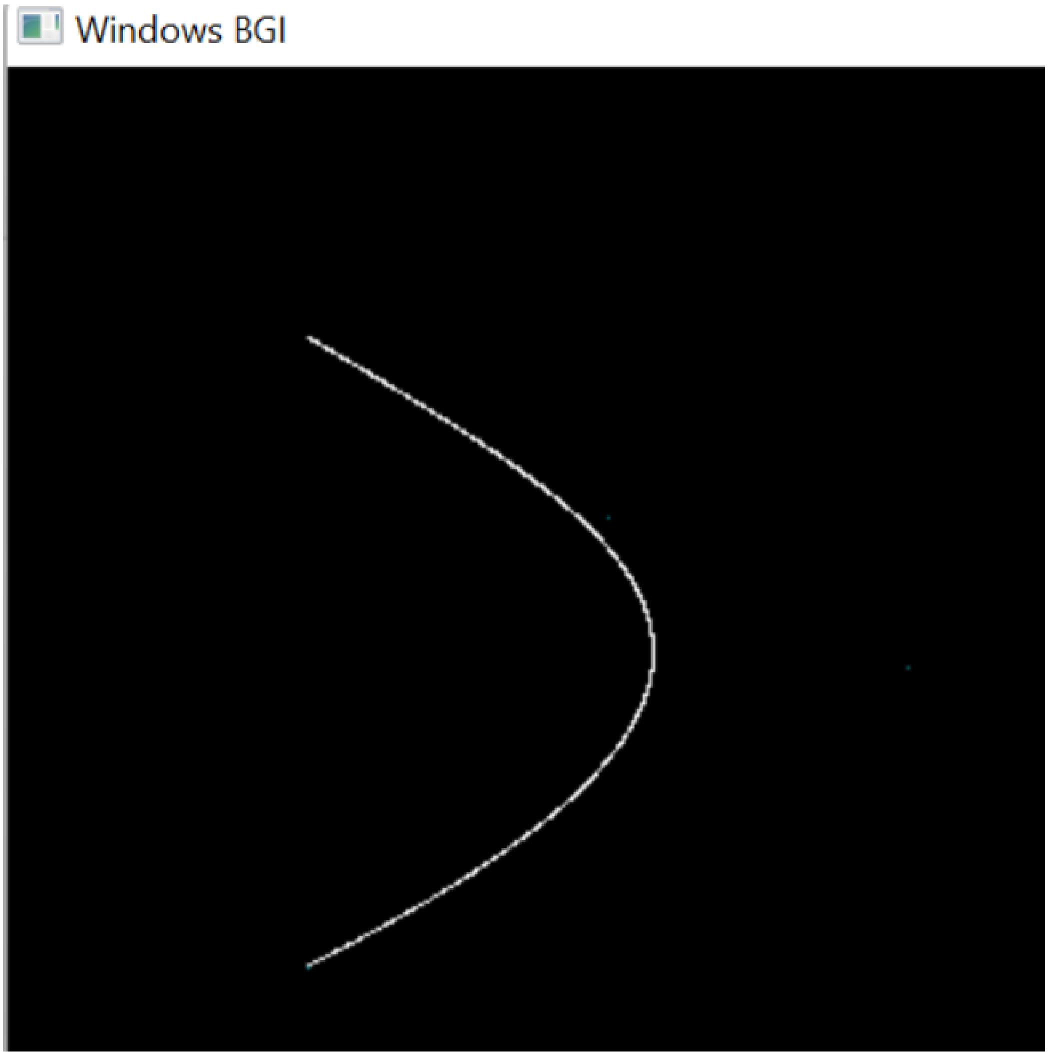
Ques 8 – Write a program to draw hermite/Bezier curve.

a) Bezier curve

Code :

```
#include<graphics.h>

#include<math.h>

#include<conio.h>

#include<stdio.h>

int main()

{

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

double put_x,put_y,t;

int gr=DETECT,gm;

initgraph(&gr,&gm,NULL);

printf("\n****** Bezier Curve ***********");

printf("\n Please enter x and y coordinates ");

for(i=0;i<4;i++)

{

scanf("%d%d",&x[i],&y[i]);

putpixel(x[i],y[i],3);              // Control Points

}


for(t=0.0;t<=1.0;t=t+0.001)          // t always lies between 0 and 1

{
```

```
put_x = pow(1-t,3)*x[0] + 3*t*pow(1-t,2)*x[1] + 3*t*t*(1-t)*x[2] + pow(t,3)*x[3]; // Formula to draw
curve

put_y =  pow(1-t,3)*y[0] + 3*t*pow(1-t,2)*y[1] + 3*t*t*(1-t)*y[2] + pow(t,3)*y[3];

putpixel(put_x,put_y, WHITE);           // putting pixel

}

getch();

closegraph();


return 0;

}
```



b) Hermite Curve

Code :

```
#include <iostream>

#include <graphics.h>

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

using namespace std;


struct point

{

  int x, y;

};


void hermite(point p1, point p4, double r1, double r4)

{

  float x, y, t;

  for (t = 0.0; t <= 1.0; t += 0.001)

  {

    x = (2 * t * t * t - 3 * t * t + 1) * p1.x + (-2 * t * t * t + 3 * t * t) * p4.x + (t * t * t - 2 * t * t + t) * r1 +
(t * t * t - t * t) * r4;

    y = (2 * t * t * t - 3 * t * t + 1) * p1.y + (-2 * t * t * t + 3 * t * t) * p4.y + (t * t * t - 2 * t * t + 1) * r1 +
(t * t * t - t * t) * r4;

    putpixel(x, y, YELLOW);

  }

}


int main()

{

  /* request auto detection */

  int gdriver = DETECT, gmode, errorcode;
```

```
  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, NULL);

  /* read result of initialization */
  errorcode = graphresult();

  /* an error occurred */
  if (errorcode != grOk)
  {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);
  }

  double r1, r4;
  point p1, p2;
  cout << "Enter 2 hermite points: " << endl;
  cin >> p1.x >> p1.y >> p2.x >> p2.y;
  cout << "Enter tangents at p1 and p4: " << endl;
  cin >> r1 >> r4;
  hermite(p1, p2, r1, r4);
  putpixel(p1.x, p1.y, WHITE);
  putpixel(p2.x, p2.y, WHITE);
  getch();
  closegraph();

  return 0;
}
```

Output :