Name : Ujjawal kumar

College roll no. :20201441

Exam roll no. :20020570038

Subject : Computer Graphics

Course : BSc. (H) Computer Science

Semester : 6th Sem, 3rd Year

Submitted to :Arun Sir

Date :30th APRIL,2023

(Ramanujan College)

Ques 1: Write a program to implement Bresenhams line drawing algorithm.

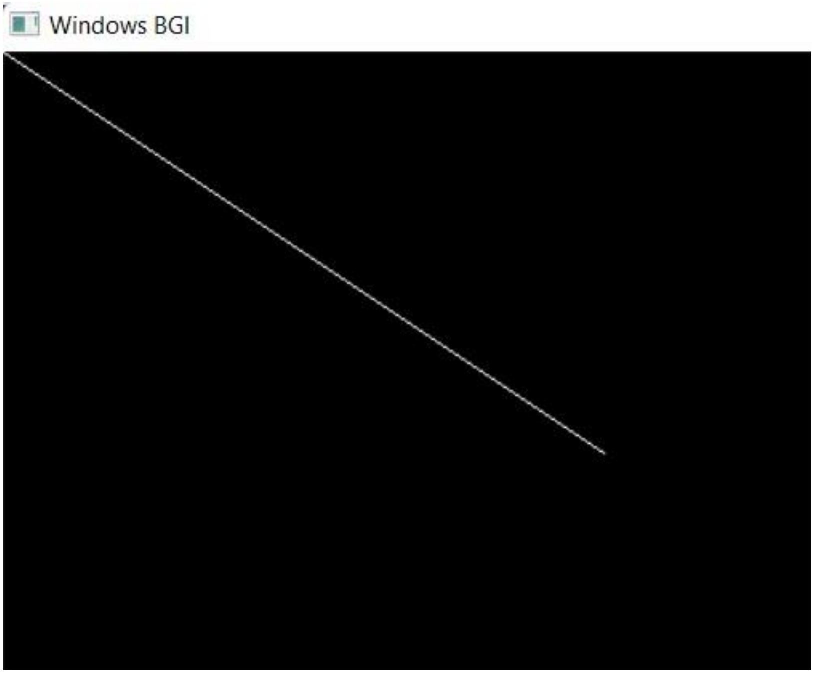
Code

```
#include <iostream>

#include <graphics.h>

#include <cstdlib>

#include <cmath>


using namespace std;


void bldaLine(int x0, int y0, int x1, int y1, int color) {
  // If the start and end points are same
  if (x0 == x1 && y0 == y1) {
    putpixel(x0, y0, color);
  } else {
    int dx = x1 - x0;
    int dy = y1 - y0;

    float m = float(dy) / float(dx);
    if (0 < m && m < 1) {
      int d = 2 * dy - dx;            // Initial value of d
      int incr_E = 2 * dy;            // Increment used for move to E
      int incr_NE = 2 * (dy - dx);    // Increment used for move to NE

      int x = x0;
      int y = y0;

      putpixel(x, y, color);          // The start pixel

      while (x < x1) {
```

```
        if (d <= 0) {                  // Choose E
          d += incr_E;
          x++;
        } else {
          d += incr_NE;
          x++;
          y++;
        }
        putpixel(x, y, color);
      }
    } else {
      cout << "\nThe slope must be between 0 and 1!\nCurrent slope: " << m;
      exit(1);
    }
  }
}

int main(void) {
 /* int x0, y0, x1, y1;
  cout << "Enter the left endpoint (x0, y0): ";
  cin >> x0 >> y0;
  cout << "Enter the right endpoint (x1, y1): ";
  cin >> x1 >> y1;
*/
  int gd = DETECT, gm, color;
  initgraph(&gd, &gm, NULL);
  bldaLine(0,0,300,200, WHITE);

  delay(10e8);
  closegraph();
```

```
  return 0;
}
```

Output



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

Code

```
#include <iostream>
#include <graphics.h>
#include <cstdlib>
#include <cmath>

using namespace std;

// Circle Points
void putpixelCirclePoints(int x, int y, int o_x, int o_y, int color) {
  putpixel(x + o_x, y + o_y, color);
```

```
  putpixel(y + o_y, x + o_x, color);
  putpixel(y + o_y, -x + o_x, color);
  putpixel(x + o_x, -y + o_y, color);
  putpixel(-x + o_x, -y + o_y, color);
  putpixel(-y + o_y, -x + o_x, color);
  putpixel(-y + o_y, x + o_x, color);
  putpixel(-x + o_x, y + o_y, color);
}

void bcdaLine(int o_x, int o_y, int radius, int color) {
  int x = 0;
  int y = radius;
  int d = 1 - radius;

  putpixelCirclePoints(x, y, o_x, o_y, color);

  while (y > x) {
    if (d < 0) {            // Select E
      d += 2 * x + 3;
    } else {                // Select SE
      d += 2 * (x - y) + 5;
      y--;
    }
    x++;
    putpixelCirclePoints(x, y, o_x, o_y, color);
  }
}

int main(void) {
 /* int o_x, o_y;     // origin (x, y)
```

```
  int radius;
  cout << "Enter the center of the circle (x, y): ";
  cin >> o_x >> o_y;
  cout << "Enter the radius of the circle: ";
  cin >> radius;
*/
  int gd = DETECT, gm, color;
  initgraph(&gd, &gm, NULL);
  bcdaLine(270,270,100, WHITE);


  delay(10e8);
  closegraph();


  return 0;
}
```

Output

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

Code

```
#include <iostream>

#include <stdio.h>

#include <graphics.h>

#include <vector>


using namespace std;


typedef unsigned int outcode;

enum _boolean {_false, _true};

enum {

  _top = 0x1,

  _bottom = 0x2,

  _right = 0x4,

  _left = 0x8

};


// Coumputing the outode

outcode compoutcode(double _x, double _y, double _xmin, double _xmax, double _ymin, double _ymax) {

  outcode code = 0;

  if (_y > _ymax)

    code |= _top;

  else if (_y < _ymin)

    code |= _bottom;

  else if (_x > _xmax)

    code |= _right;

  else if (_x < _xmin)
```

```
    code |= _left;

  return code;
}

/* Cohen-Sutherland clipping algorithm for line P0 = x0, y0 to P1 = (x1, y1) and
 * clip rectangle with diagonal from (xmin, ymin) to (xmax, ymax)
 */
void cohen_sutherland_line_clip(double _x0, double _y0, double _x1, double _y1, double _xmin, double _xmax, double _ymin, double _ymax) {
  // Outcodes for P0, P1, and whatever point lies outside the clip rectangle
  outcode outcode0, outcode1, outcodeOut;
  _boolean accept = _false, done = _false;
  outcode0 = compoutcode(_x0, _y0, _xmin, _xmax, _ymin, _ymax);
  outcode1 = compoutcode(_x1, _y1, _xmin, _xmax, _ymin, _ymax);

  do {
    if (!(outcode0 | outcode1)) {
      accept = _true;
      done = _true;
    } else if (outcode0 & outcode1) {
      done = _true;
    } else {
      /* Failed both tests, so calculate the line segment to clip
       * from an outside to an intersection with clip edge.
       */
      double x, y;

      // At least one endpoint is outside the clip rectangle; pick it.
      outcodeOut = outcode0 ? outcode0 : outcode1;
```

```
/* Now finding the intersection point;
 * using formulas y = y0 + slope * (x - x0),
 *              x = x0 + (1/slope) * (y - y0)
 */
if (outcodeOut & _top) {              // Divide line at top of clip rectangle
  x = _x0 + (_x1 - _x0) * (_ymax - _y0) / (_y1 - _y0);
  y = _ymax;
} else if (outcodeOut & _bottom) {      // Divide line at bottom of clip rectangle
  x = _x0 + (_x1 - _x0) * (_ymin - _y0) / (_y1 - _y0);
  y = _ymin;
} else if (outcodeOut & _right) {       // Divide line at right of clip rectangle
  y = _y0 + (_y1 - _y0) * (_xmax - _x0) / (_x1 - _x0);
  x = _xmax;
} else {                                // Divide line at top of clip rectangle
  y = _y0 + (_y1 - _y0) * (_xmin - _x0) / (_x1 - _x0);
  x = _xmin;
}

/* Now we move outside point to intersection point to clip,
 * and get ready for the next pass.
 */
if (outcodeOut == outcode0) {
  _x0 = x;
  _y0 = y;
  outcode0 = compoutcode(_x0, _y0, _xmin, _xmax, _ymin, _ymax);
} else {
  _x1 = x;
  _y1 = y;
  outcode1 = compoutcode(_x1, _y1, _xmin, _xmax, _ymin, _ymax);
}
```

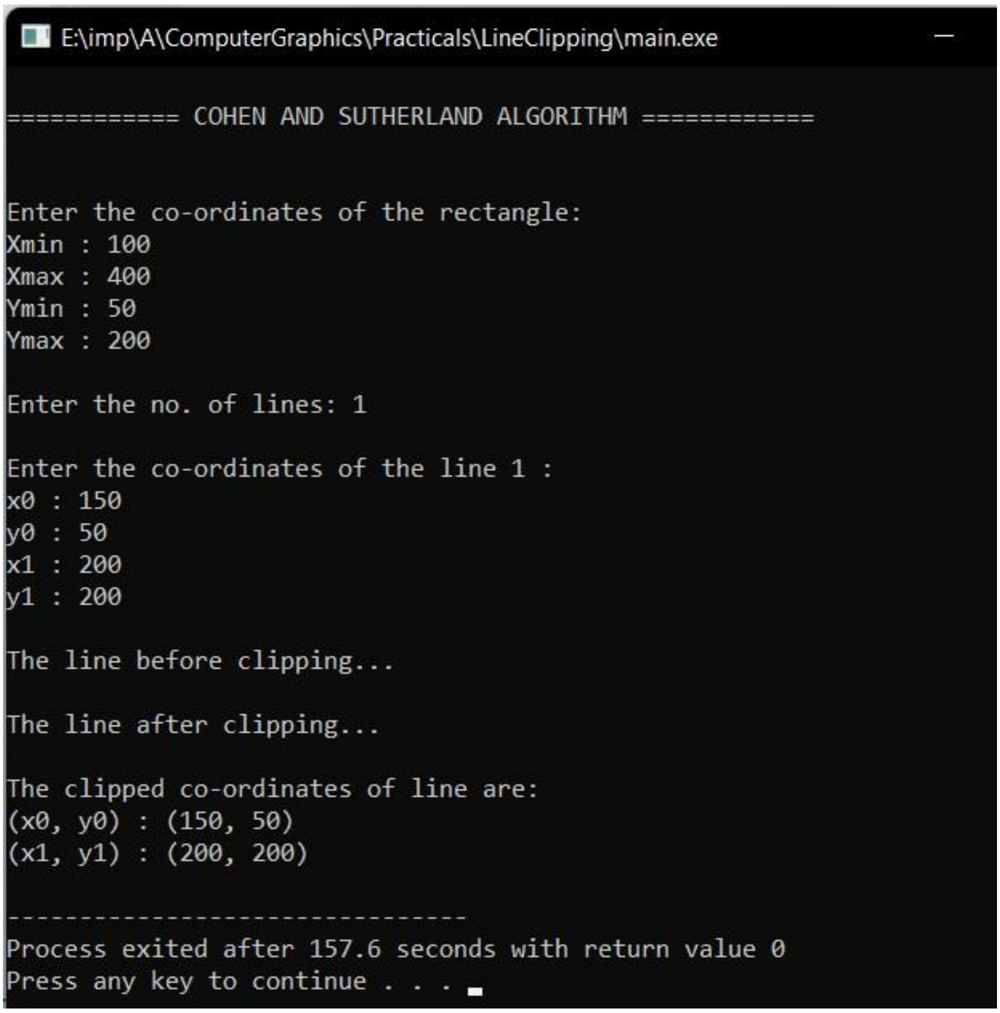
```
    }     // Subdivide
  } while (done == _false);

  if (accept) {
    line(_x0, _y0, _x1, _y1);
    cout << "\nThe clipped co-ordinates of line are:"
        << "\n(x0, y0) : (" << _x0 << ", " << _y0 << ")"
        << "\n(x1, y1) : (" << _x1 << ", " << _y1 << ")"
        << endl;
  }
}

int main() {
            cout << "\n============ COHEN AND SUTHERLAND ALGORITHM
============\n";
  int x0, y0, x1, y1;
  int xmin, xmax, ymin, ymax;
  int lines_count;
  vector<vector<int> > lines;
  vector<int> point;

  int gd = DETECT, gm;
  initgraph(&gd, &gm, NULL);
  setbkcolor(RED);

  cout << "\n\nEnter the co-ordinates of the rectangle:";
  cout << "\nXmin : ";
  cin >> xmin;
  cout << "Xmax : ";
  cin >> xmax;
  cout << "Ymin : ";
```

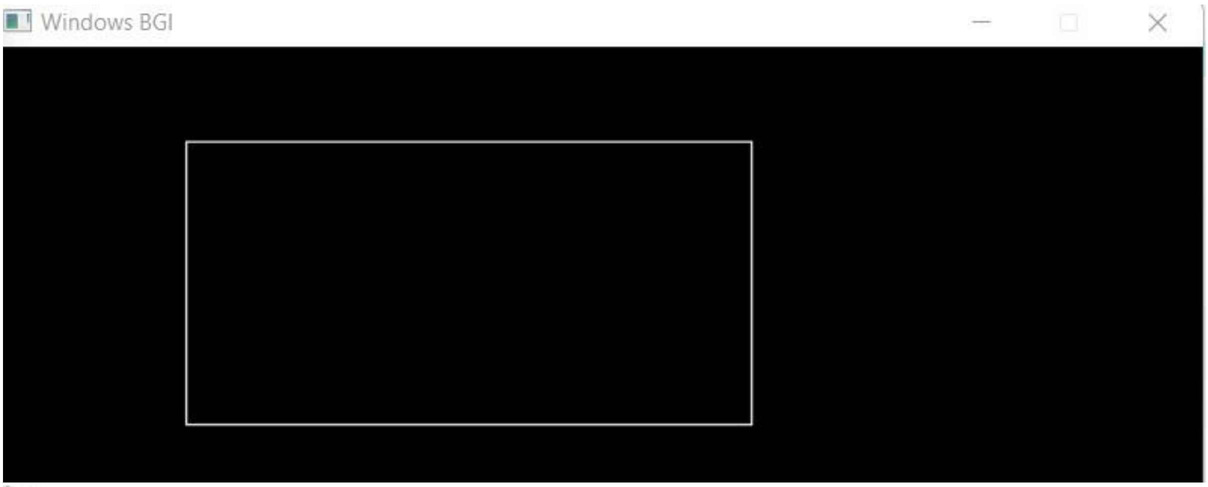
```
cin >> ymin;
cout << "Ymax : ";
cin >> ymax;

rectangle(xmin, ymin, xmax, ymax);

cout << "\nEnter the no. of lines: ";
cin >> lines_count;

for (int i = 0; i < lines_count; i++) {
  cout << "\nEnter the co-ordinates of the line " << i+1 << " :";
  cout << "\nx0 : ";
  cin >> x0;
  cout << "y0 : ";
  cin >> y0;
  cout << "x1 : ";
  cin >> x1;
  cout << "y1 : ";
  cin >> y1;

  point.push_back(x0);
  point.push_back(y0);
  point.push_back(x1);
  point.push_back(y1);

  lines.push_back(point);

  point.clear();
}
```

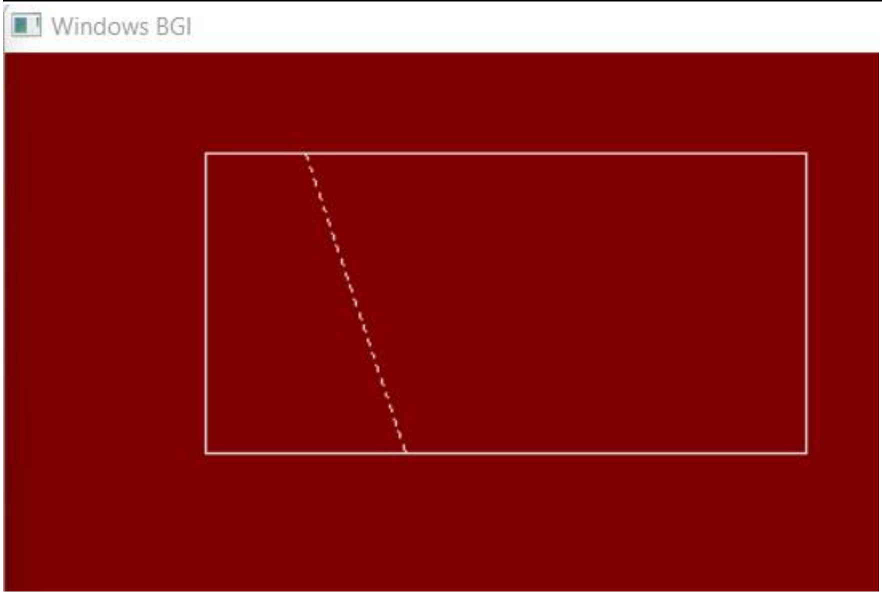
```
// The entered co-ordinates of lines
// for (int i = 0; i < lines_count; i++) {
//   for (int j = 0; j < 4; j++) {
//     cout << lines[i][j] << " ";
//   }
//   cout << endl;
// }

cout << "\nThe line before clipping...\n";
for (int i = 0; i < lines_count; i++) {
  x0 = lines[i][0];
  y0 = lines[i][1];
  x1 = lines[i][2];
  y1 = lines[i][3];
  line(x0, y0, x1, y1);
}

delay(3000);
cleardevice();
delay(200);

cout << "\nThe line after clipping...\n";
rectangle(xmin, ymin, xmax, ymax);
setlinestyle(DOTTED_LINE, 1, 1);

for (int i = 0; i < lines_count; i++) {
  x0 = lines[i][0];
  y0 = lines[i][1];
  x1 = lines[i][2];
  y1 = lines[i][3];
```

```
    cohen_sutherland_line_clip(x0, y0, x1, y1, xmin, xmax, ymin, ymax);
  }


  getch();
  return 0;
}
```

Output




Windows BGI


Windows BGI

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

Code

```
#include <iostream>

#include <stdio.h>

#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;

}


void draw_polygon(int size, int points[][2]) {

	int i;

	for (i = 0; i < size - 1; i+=1) {

   line(points[i][0], points[i][1], points[i+1][0], points[i+1][1]);

  }
```

```
        line(points[i][0], points[i][1], points[0][0], points[0][1]);
}

// Clipping 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) {
```

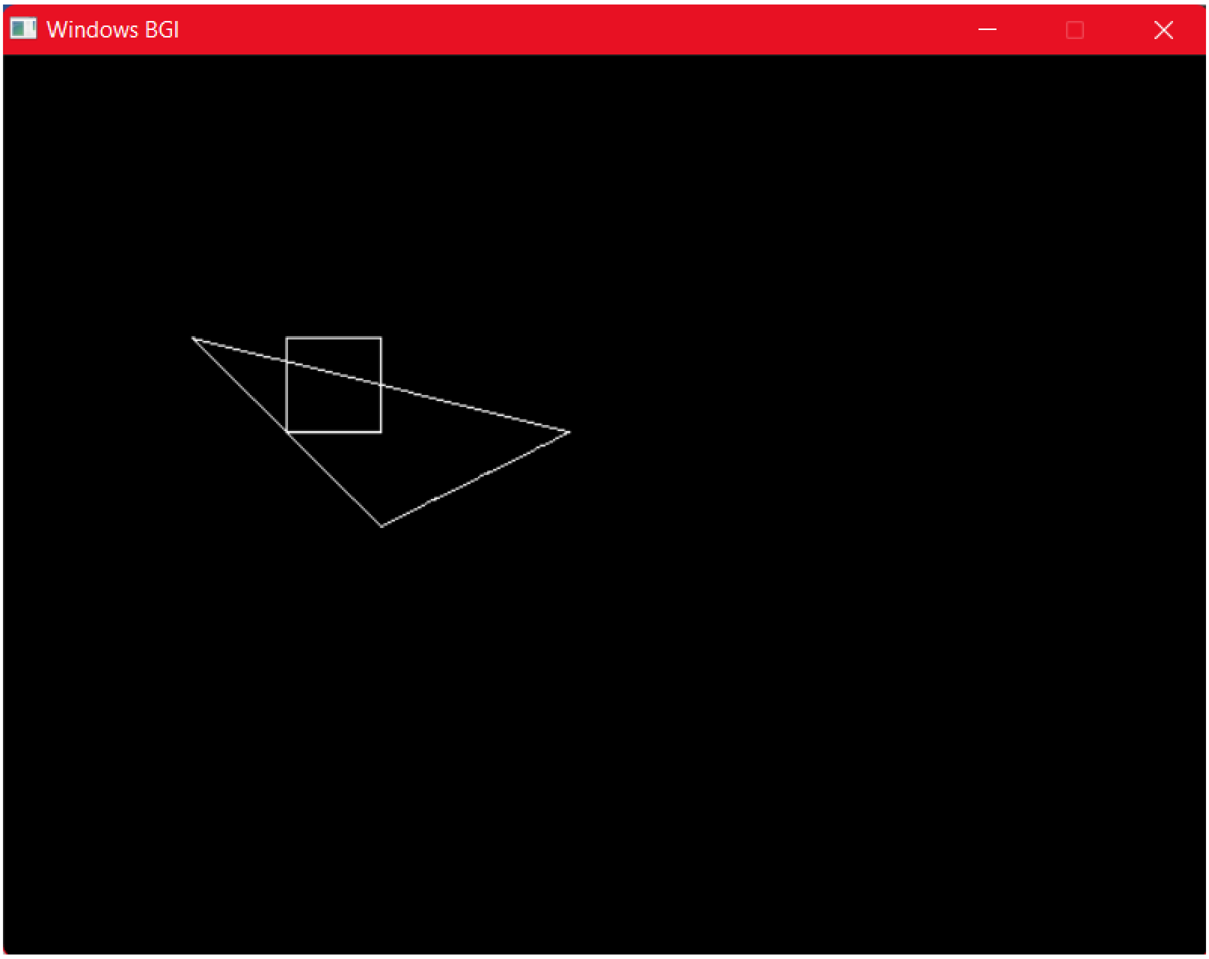
```
            // Point of intersection with edge and the second point is added
            new_points[new_poly_size][0] = x_intersect(x1, y1, x2, y2, ix, iy, kx,
ky);
            new_points[new_poly_size][1] = y_intersect(x1, y1, x2, y2, ix, iy, kx,
ky);
            new_poly_size++;

            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];
```

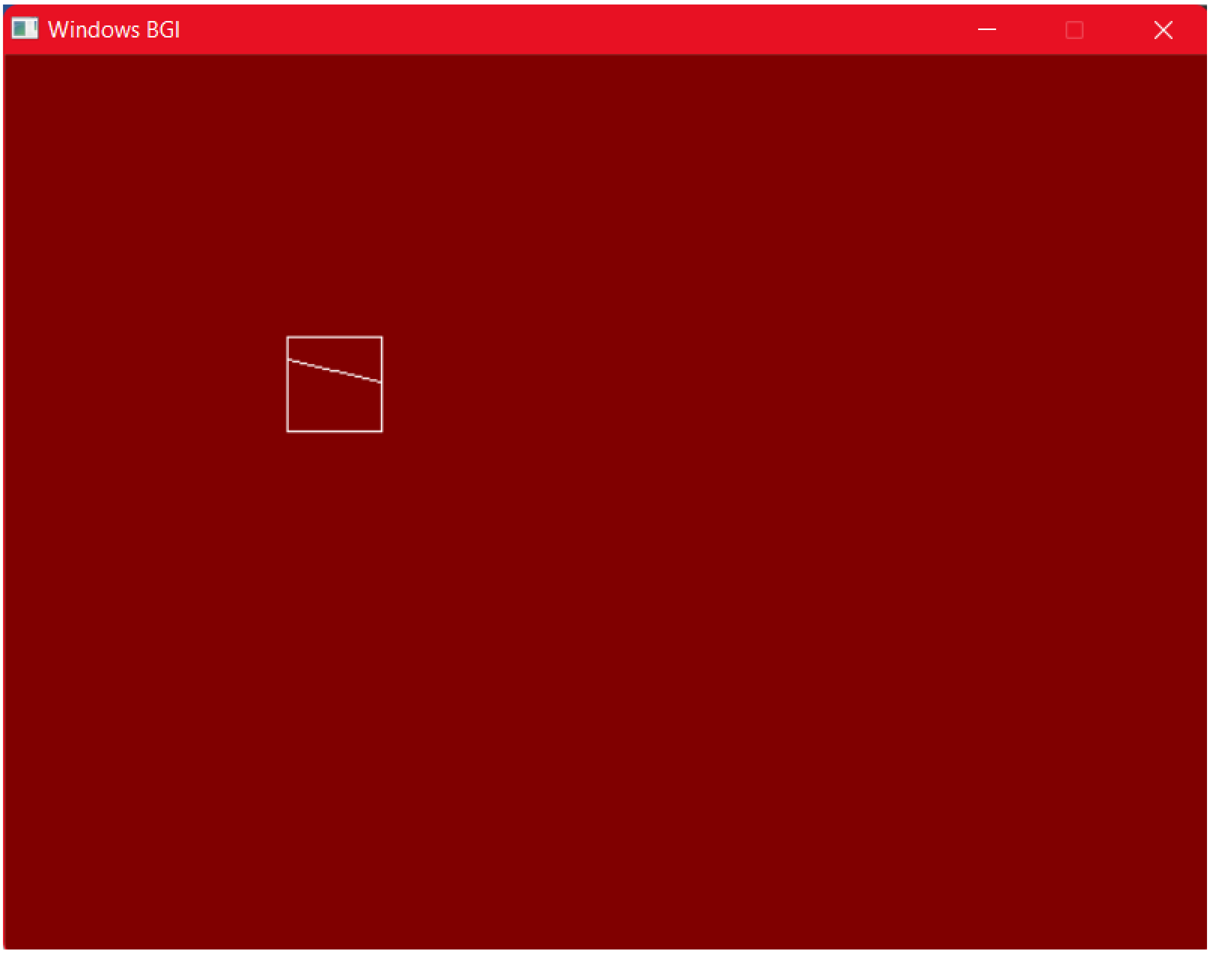
```
                    poly_points[i][1] = new_points[i][1];
          }
}

// Implements Sutherlandâ€“Hodgman algorithm
void suthHodgClip(int poly_points[][2], int poly_size, int clipper_points[][2], int
clipper_size) {
          int i, k;
          for (i=0; i<clipper_size; i++) {
                    k = (i+1) % clipper_size;

                    // Passing 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 << "\nVertices of the clipped polygon are :";
          for (i=0; i < poly_size; i++) {
                    cout << "\n(x" << i << ", y" << i << ") : ("
                                                  << poly_points[i][0] << ", " << poly_points[i][1] << ")
";
          }
          draw_polygon(poly_size, poly_points);
}

int main() {
          cout << "\n=========== SUTHERLAND HODGEMAN ALGORITHM
===========\n";
```

```
	int i;
int x0, y0, x1, y1;

	int poly_size;
	cout << "\nEnter the size of polygon : ";
	cin >> poly_size;
	int poly_points[MAX_POINTS][2];

	int clipper_size;
	cout << "\nEnter the size of clipper : ";
	cin >> clipper_size;
	int clipper_points[MAX_POINTS][2];

	cout << "\nEnter the points of polygon : \n";
	for (i = 0; i < poly_size; i++) {
		cout << "Enter the point " << i+1 << " : ";
		cin >> poly_points[i][0] >> poly_points[i][1];
	}

	cout << "\nEnter the points of clipper : \n";
	for (i = 0; i < clipper_size; i++) {
		cout << "Enter the point " << i+1 << " : ";
		cin >> clipper_points[i][0] >> clipper_points[i][1];
	}

int gd = DETECT, gm;
initgraph(&gd, &gm, NULL);
setbkcolor(RED);

	draw_polygon(poly_size, poly_points);
```

```
        draw_polygon(clipper_size, clipper_points);

  delay(3000);
  cleardevice();
  delay(200);

        draw_polygon(clipper_size, clipper_points);

        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 fill 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
```

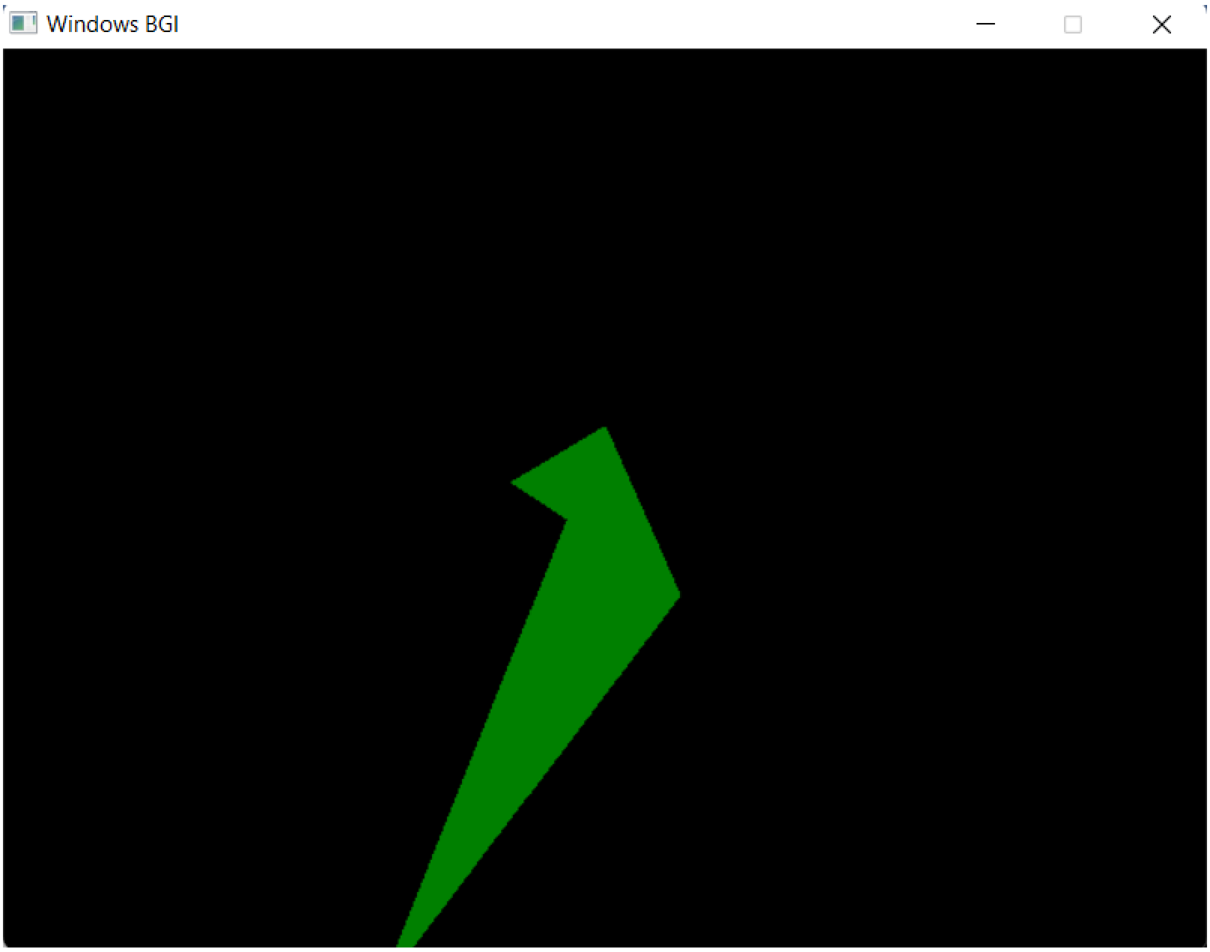
```
{
edge = (Edge *) malloc (sizeof(Edge));
if (v1.y < v2.y) // upgoing edge
makeEdgeRec(v1, v2, yNext(i, cnt, pts), edge, edges);
else //down-going edge
makeEdgeRec(v2, v1 , yPrev, edge, edges);
}
yPrev = v1.y;
v1 = v2;
}
}
void buildActiveList(int scan, Edge *active, Edge *edges[])
{
Edge *p, *q;
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 transformations on a 2D object (use homogenous coordinates).

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],Y
ELLOW);
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}};
```

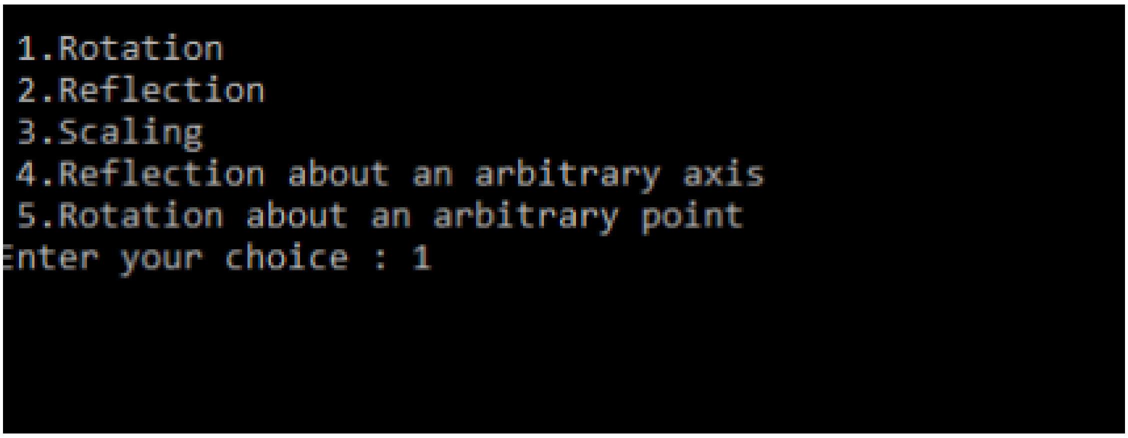
```
int t[2][2]={{5,0},{0,5}};
for( i = 0 ; i < 3 ; i++){
for(j=0 ; j<2 ; j++){
c[i][j]=0;} }
dda_line(a[0][0],a[0][1],a[1][0],a[1][1],YELLOW);
dda_line(a[1][0],a[1][1],a[2][0],a[2][1],YELLOW);
dda_line(a[2][0],a[2][1],a[0][0],a[0][1],YELLOW);
for ( i=0;i<3;i++){
for ( j=0;j<2;j++){
for ( k=0;k<2;k++){
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];
```

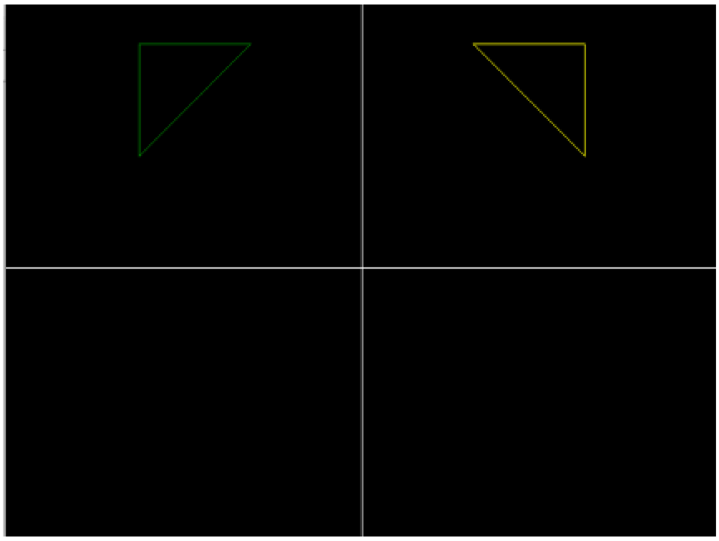
```
} } }
void reflection_arbitrary(){
int xmid , ymid;
xmid = getmaxx()/2;
ymid = getmaxy()/2;
line(xmid , 0 , xmid , getmaxy());
line(0 , ymid , getmaxx() , ymid);
int a[3][3]={{200,200,1},{200,100,1},{100,200,1}};
int t[3][3]={{1,0,0},{0,1,0},{0,0,1}};
int r[3][3]={{-1,0,0},{0,-1,0},{0,0,1}};
int 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);
```

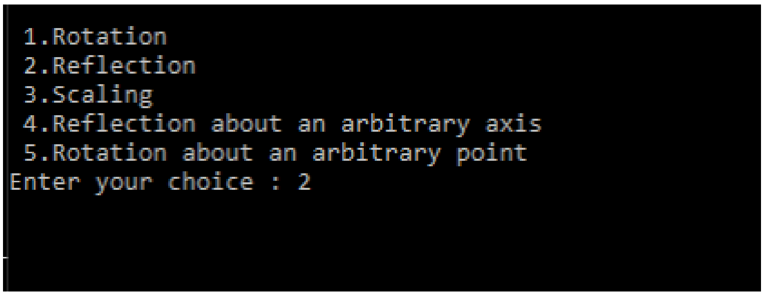
```
int c[3][3] , i , j , k;
int l[1][3]={{200,200,1}};
int a[3][3]={{200,200,1},{200,100,1},{100,200,1}};
int t[3][3]={{1,0,0},{0,1,0},{-133,-133,1}};
int r[3][3]={{-1,0,0},{0,-1,0},{0,0,1}};
int tinv[3][3]={{1,0,0},{0,1,0},{133,133,1}};
dda_line(a[0][0],a[0][1],a[1][0],a[1][1],YELLOW);
dda_line(a[1][0],a[1][1],a[2][0],a[2][1],YELLOW);
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";
```

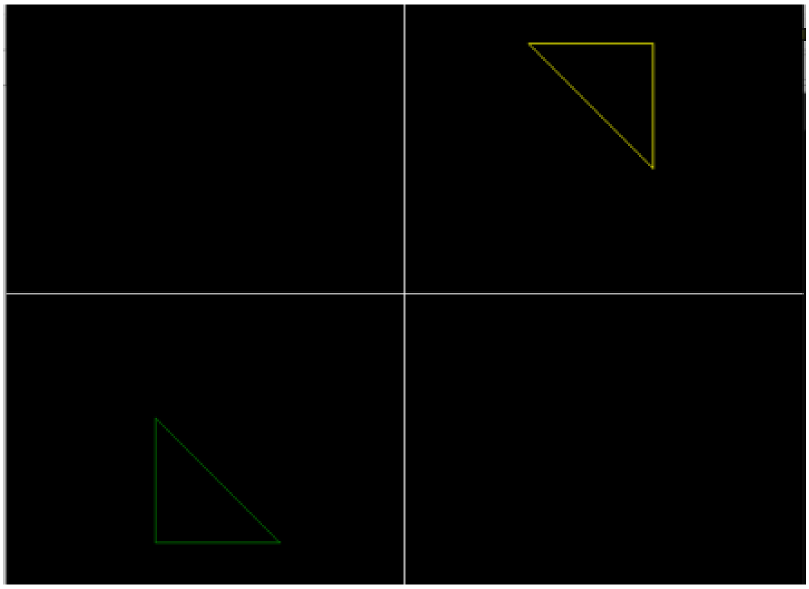
```
cout<<"Enter your choice : ";
cin>>n;switch(n){
case 1 : rotate();
break;
case 2 : reflection();
break;
case 3 : scaling();
break;
case 4 : reflection_arbitrary();
break;
case 5 : rotation_arbitrary();
break;
default : cout<<"Invalid Choice\n";
}
getch();
}
```

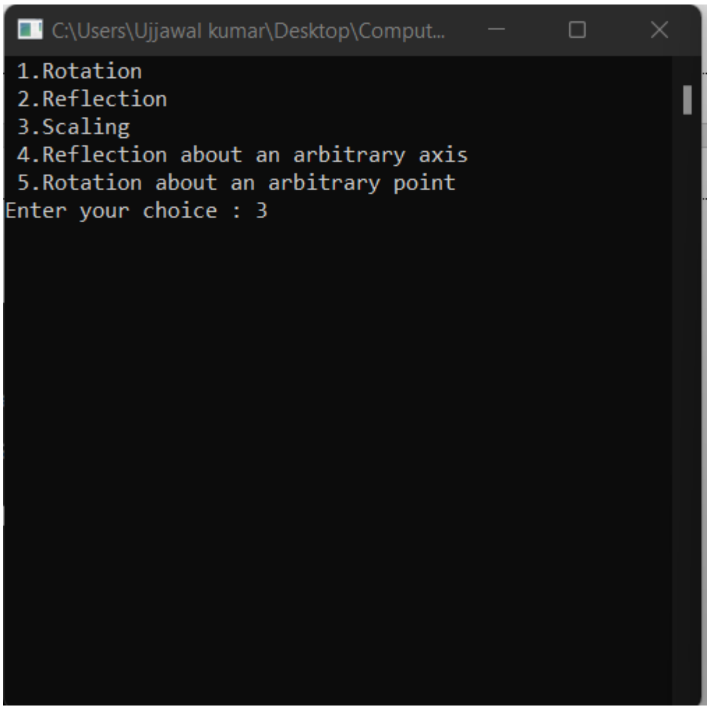
Output

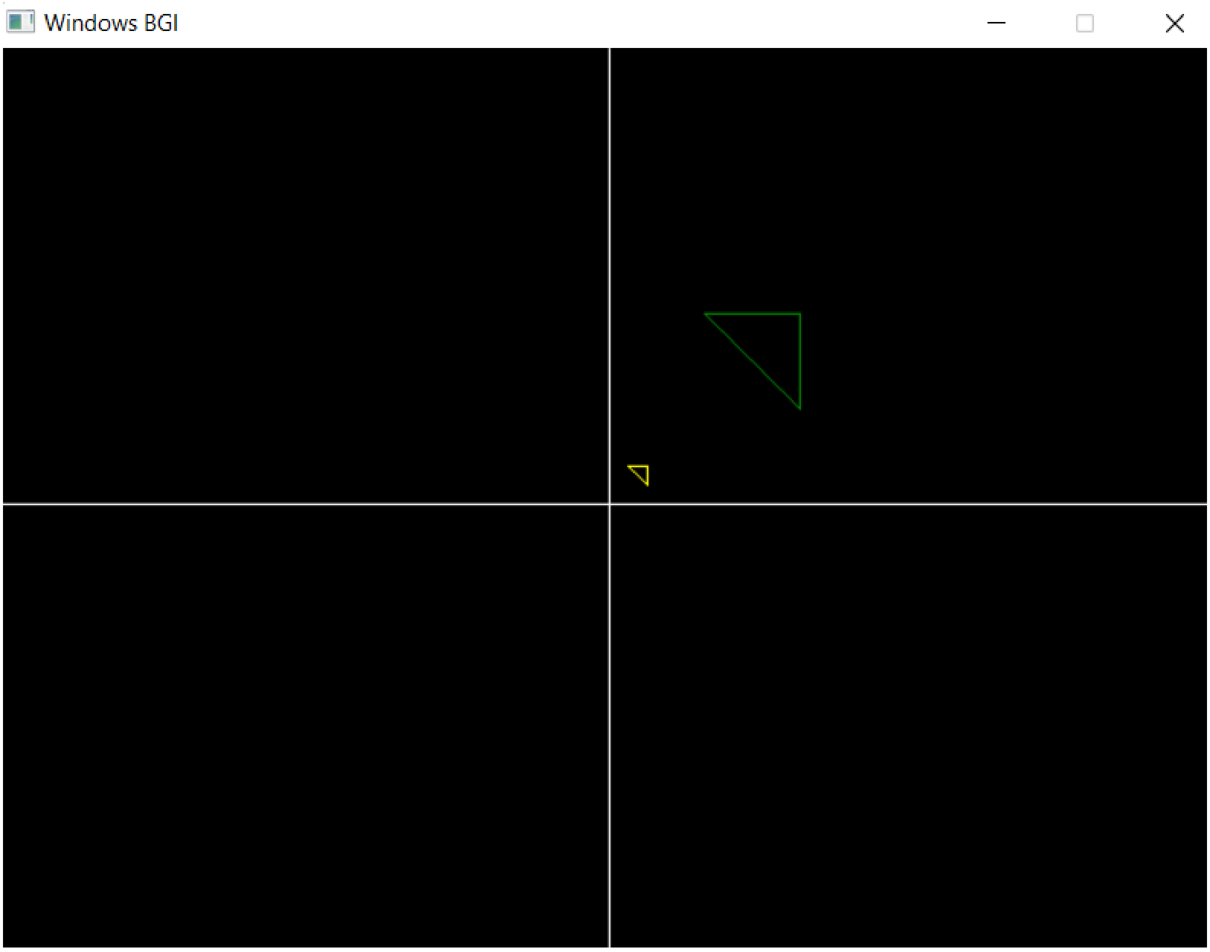


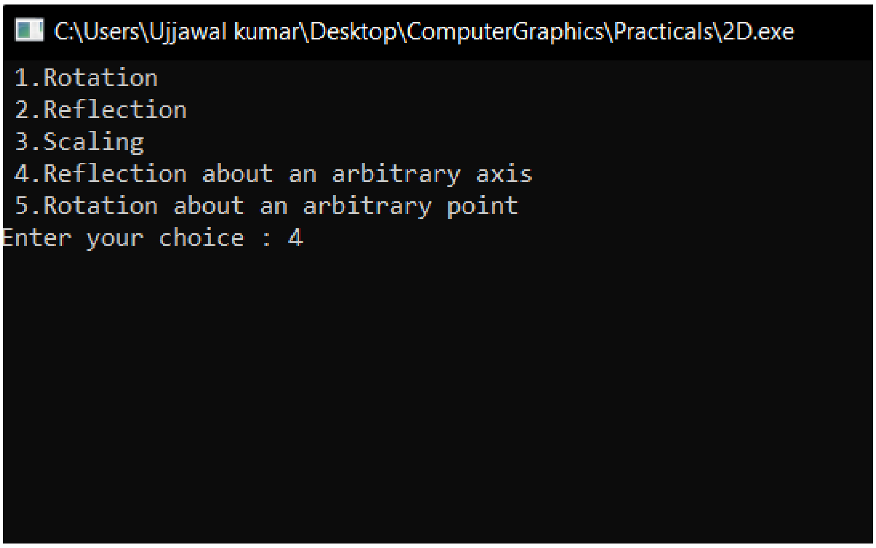


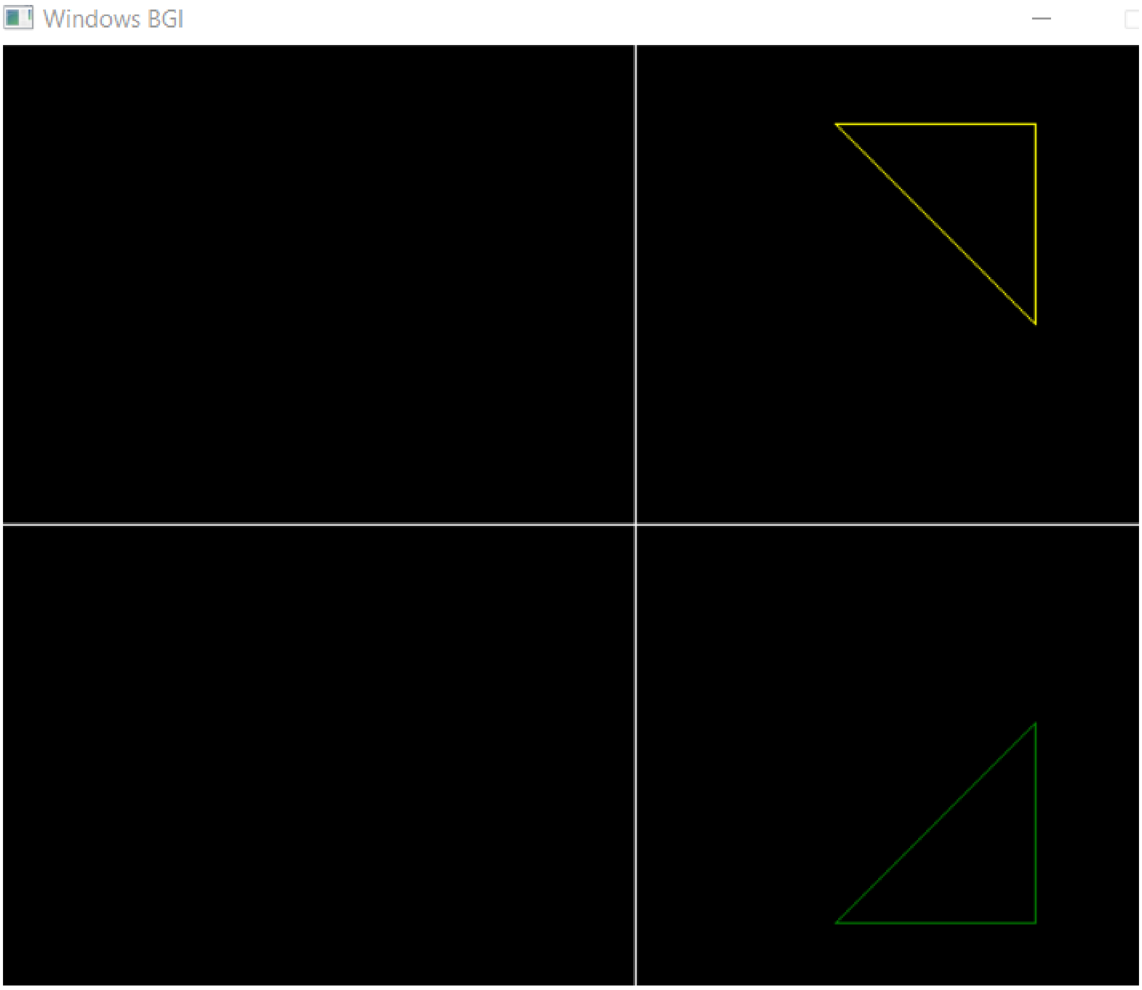

1.Rotation
2.Reflection
3.Scaling
4.Reflection about an arbitrary axis
5.Rotation about an arbitrary point
Enter your choice : 2

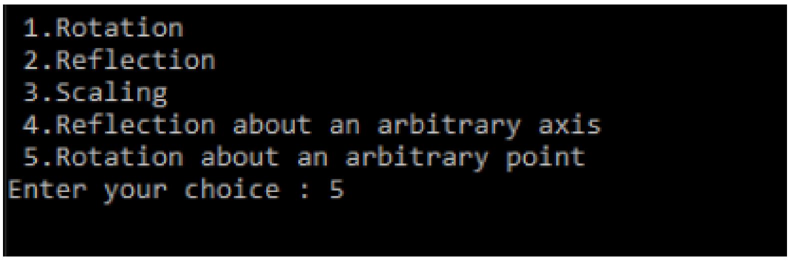


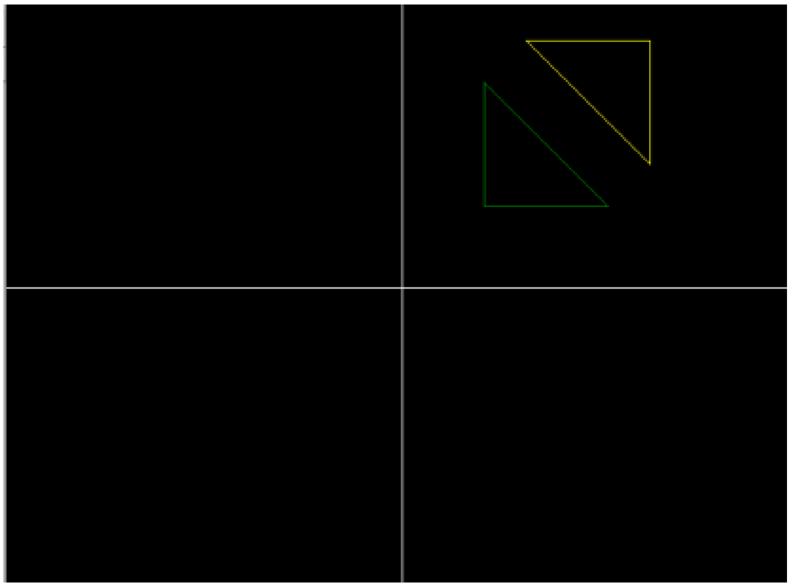

C:\Users\Ujjawal kumar\Desktop\Comput...
1.Rotation
2.Reflection
3.Scaling
4.Reflection about an arbitrary axis
5.Rotation about an arbitrary point
Enter your choice : 3


Windows BGI


C:\Users\Ujjawal kumar\Desktop\ComputerGraphics\Practicals\2D.exe
1.Rotation
2.Reflection
3.Scaling
4.Reflection about an arbitrary axis
5.Rotation about an arbitrary point
Enter your choice : 4


Windows BGI





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

Code

```
#include<iostream>

#include<dos.h>

#include<stdio.h>

#include<math.h>

#include<conio.h>

#include<graphics.h>

#include<process.h>


using namespace std;
```

```
void draw_cube(double edge[20][3]){
        double x1,x2,y1,y2;
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];
```

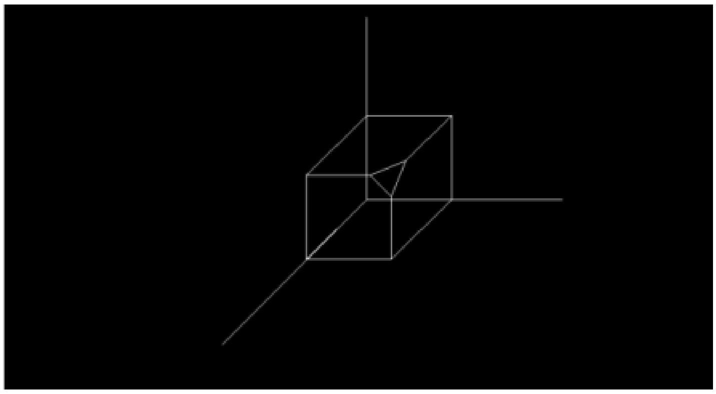
```
edge[i][2]=-edge[i][2];
}
draw_cube(edge);
break;
case 3: for(i=0;i<20;i++){
edge[i][2]=edge[i][2];
edge[i][0]=-edge[i][0];
edge[i][1]=-edge[i][1];
}
draw_cube(edge);
break;
}}
void perspect(double edge[20][3]){
int n;
int i;
double p,q,r;
cleardevice();
cout<<" 1.X-Axis \n 2.Y-Axis \n 3.Z-Axis\n";
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;
} }
int main(){

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";
```

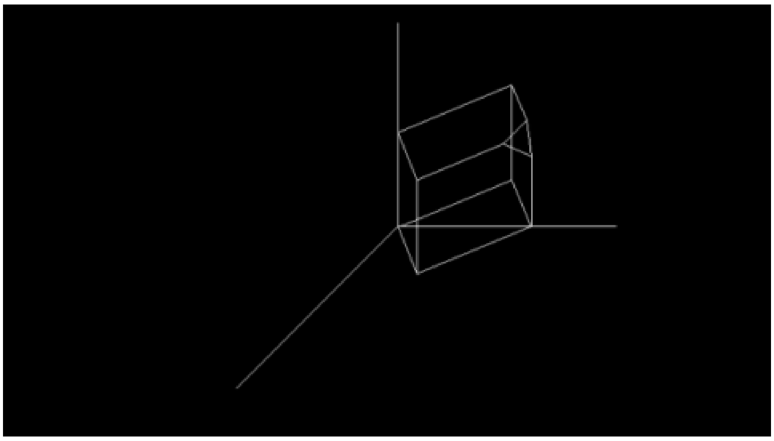
```
cout<<" Enter Your Choice : ";cin>>n;
switch(n){
case 1: draw_cube(edge);
break;
case 2: rotate(edge);
break;
case 3: reflect(edge);
break;
case 4: translate(edge);
break;
case 5: perspect(edge);
break;
default: cout<<" Invalid Choice\n ";
}
getch();
return 0;
}
```

Output

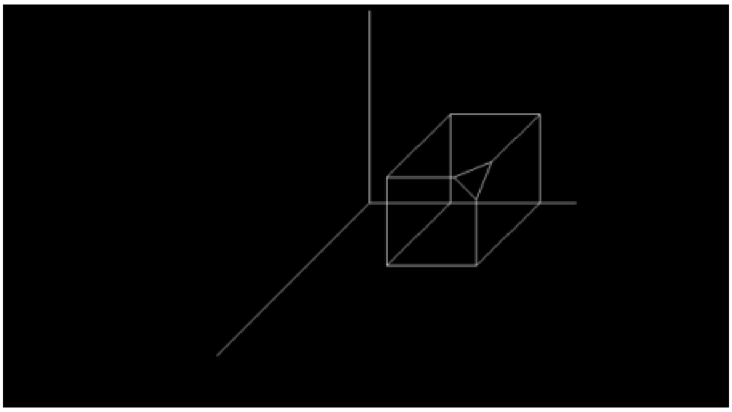
ORIGINAL CUBE:



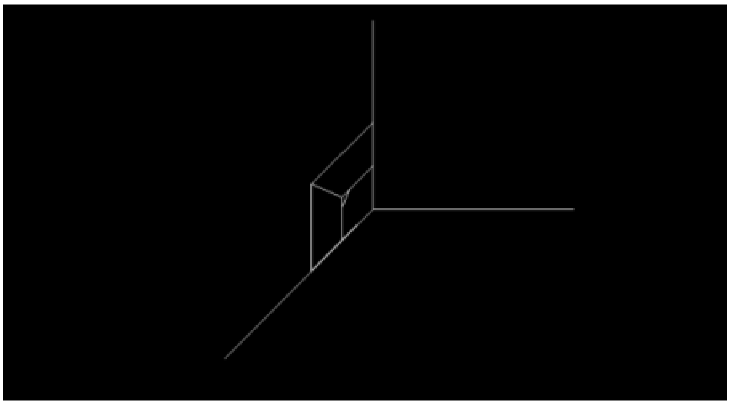
ROTATION ABOUT Y-AXIS BY AN ANGLE OF 45 DEGREE:



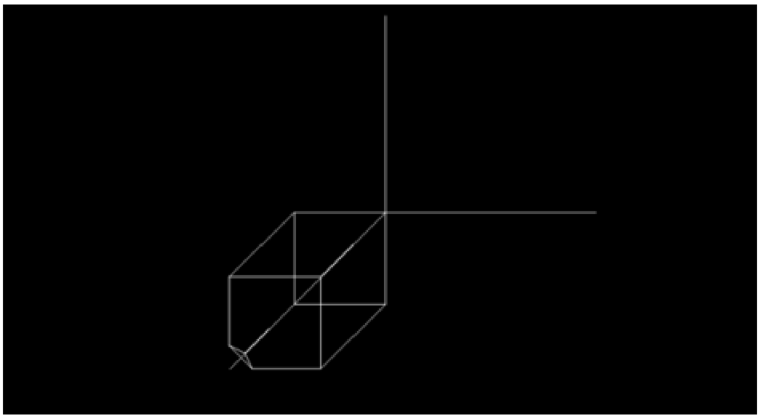
TRANSLATION FACTORS AS 20, 30, 40:



PERSPECTIVE PROJECTION ABOUT X-AXIS WHEN P=50:



REFLECTION ABOUT Z-AXIS:



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

Code

```
#include <iostream>
#include <stdio.h>
#include <graphics.h>
#include <vector>
#include <cmath>

using namespace std;

#define MAX 4

void hermite_curve(vector<vector<int> > &_controlPoints) {
  int _x, _y, _x1, _y1;

  setcolor(BLACK);
  for (int i = 0; i < MAX; i++) {
    _x = _controlPoints[i][0];
    _y = _controlPoints[i][1];
    putpixel(_x, _y, BLACK);
```

```
    circle(_x, _y, 5);
  }

  setlinestyle(SOLID_LINE, 1, NORM_WIDTH);
  setcolor(RED);
  for (int i = 0; i < MAX - 2; i++) {
    _x = _controlPoints[i][0];
    _y = _controlPoints[i][1];
    _x1 = _controlPoints[i+2][0];
    _y1 = _controlPoints[i+2][1];
    line(_x, _y, _x1, _y1);
  }

  for (float t = 0; t <= 1; t += 0.0001) {
    _x = (2*pow(t, 3) - 3*pow(t, 2) + 1)*_controlPoints[0][0]
          + (-2*pow(t, 3) + 3*pow(t, 2))*_controlPoints[1][0]
          + (pow(t, 3) - 2*pow(t, 2) + t)*_controlPoints[2][0]
          + (pow(t, 3) - pow(t, 2))*_controlPoints[3][0];

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

    putpixel(_x, _y, BLACK);
    for (int i=0; i<=10e4; i++);
  }
}

void bezier_curve(vector<vector<int> > &_controlPoints) {
```

```
int _x, _y, _x1, _y1;

setcolor(BLACK);
for (int i = 0; i < MAX; i++) {
  _x = _controlPoints[i][0];
  _y = _controlPoints[i][1];
  putpixel(_x, _y, BLACK);
  circle(_x, _y, 5);
}

setlinestyle(SOLID_LINE, 1, NORM_WIDTH);
setcolor(RED);
for (int i = 0; i < MAX - 1; i++) {
  _x = _controlPoints[i][0];
  _y = _controlPoints[i][1];
  _x1 = _controlPoints[i+1][0];
  _y1 = _controlPoints[i+1][1];
  line(_x, _y, _x1, _y1);
}
// B(t) = (1-t)^3P0 + 3(1-t)^2tP1 + 3(1-t)^2t^2P2 + t^3P3
for (float t = 0; t <= 1; t += 0.0001) {
  _x = pow(t, 3)*(_controlPoints[3][0] + 3*(_controlPoints[1][0] - _controlPoints[2][0]) - _controlPoints[0][0])
       + 3*pow(t, 2)*(_controlPoints[0][0] - 2*_controlPoints[1][0] + _controlPoints[2][0])
       + 3*t*(_controlPoints[1][0] - _controlPoints[0][0])
       + _controlPoints[0][0];

  _y = pow(t, 3)*(_controlPoints[3][1] + 3*(_controlPoints[1][1] - _controlPoints[2][1]) - _controlPoints[0][1])
       + 3*pow(t, 2)*(_controlPoints[0][1] - 2*_controlPoints[1][1] + _controlPoints[2][1])
       + 3*t*(_controlPoints[1][1] - _controlPoints[0][1])
```

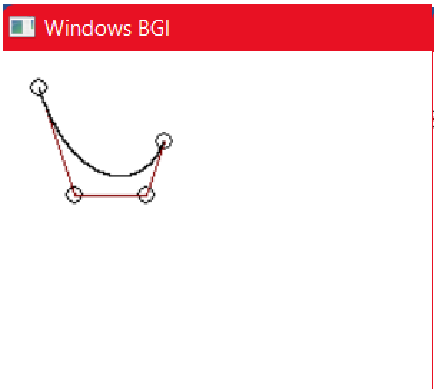
```
        + _controlPoints[0][1];

    putpixel(_x, _y, BLACK);
    for (int i=0; i<=10e4; i++);
  }
}

vector<vector<int> > input_control_points() {
  int x, y;
  vector<vector<int> > controlPoints;
  vector<int> point;

  for (int i = 0; i < MAX; i++) {
      cout << "Enter the control point " << i+1 << " (x, y) : ";
      cin >> x >> y;
      point.push_back(x);
      point.push_back(y);

      controlPoints.push_back(point);
      point.clear();
    }

    //The entered co-ordinates of ponts
    // for (int i = 0; i < MAX; i++) {
    //   for (int j = 0; j < 2; j++) {
    //     cout << controlPoints[i][j] << " ";
    //   }
    //   cout << endl;
    // }
```

```
    return controlPoints;
}

int main() {
            cout << "\n============ HERMITE AND BEZIER CURVE ============\n";

  int curveChoice;
  vector<vector<int> > controlPoints;

  int gd = DETECT, gm;
  initgraph(&gd, &gm, NULL);
  setbkcolor(WHITE);

  curve_menu:
    cleardevice();
    cout << "\n===== MENU ====="
         << "\n1. Hermite Curve"
         << "\n2. Bezier Curve"
         << "\n0. Exit" << endl;
    cout << "\nEnter your choice : ";
    cin >> curveChoice;

    switch (curveChoice) {
      case 1:
        cout << "\n====== Hermite Curve ======\n";
        controlPoints = input_control_points();
        hermite_curve(controlPoints);
        break;

      case 2:
```

```
        cout << "\n===== Bezier Curve =====\n";
        controlPoints = input_control_points();
        bezier_curve(controlPoints);
        break;

      case 0:
        cout << "\nExiting...\n";
        exit(0);

      default:
        cout << "\nINVALID CHOICE... TRY AGAIN!!!";
        break;
    }

  delay(10e3);
  goto curve_menu;

  getch();
  return 0;
}
```

Output

Bezier



Hermite