**FINAL PRACTICAL FILE**

**Name:** Ravish Ranjan

**Semester:** 6th Semester

**Course:** BSc. Hons. Computer Science

**Roll No.:** 21058570040

**Que1** Write a program to implement DDA and Bresenham’s line drawing algorithm.

**Ans**

**#DAA**

#include <graphics.h>

#include <stdio.h>

#include <dos.h>

void drawseg(int x1,int y1,int x2,int y2){

    float m = (x2-x1)/(y2-y1);

    int prex = x1;

    int prey = y1;

    int newx = x1;

    int newy = y1;

    while(prex<= x2) {

        newx += 1;

        newy = prey + m;

        putpixel(newx,newy,LIGHTGREEN);

        prex = newx;

        prey = newy;

        delay(10);

    }

}

int main(){

    int gd = DETECT,gm;

    initgraph(&gd,&gm,(char\*) "");

    setcolor(getmaxcolor());

    drawseg(100,100,300,200);

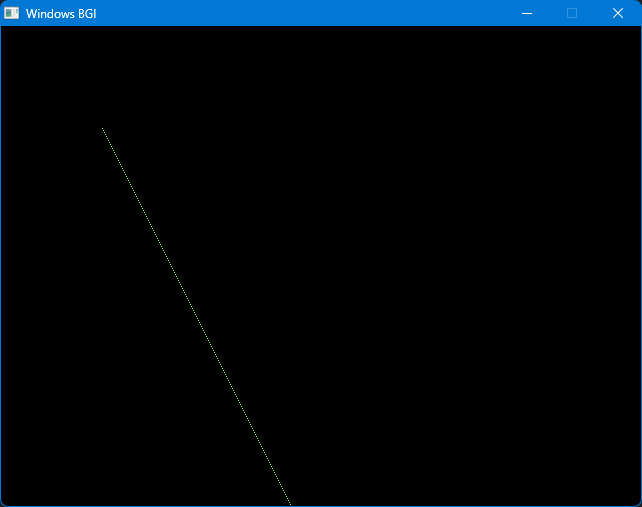
    getch();

    closegraph();

    return 0;

}

**#Output**



**#** Bresenham’s

#include <stdio.h>

#include <graphics.h>

#include <dos.h>

float getD\_T(float di,int dy,int dx){

    return di + 2\*(dy - dx);

}

float getD\_S(float di, int dx){

    return di - 2\*dx;

}

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

    float dx = x1 - x0;

    float dy = y1 - y0;

    int di = 2\*dy - dx;

    float m = dy / dx;

    int x\_old = x0;

    int y\_old = y0;

    if (m<=1){

        while (x\_old <= x1){

            putpixel(x\_old,y\_old,LIGHTBLUE);

            x\_old+=1;

            if (di>=0) {

                y\_old++;

                di = getD\_T(di,dy,dx);

            } else {

                di = getD\_S(di,dx);

            }

        delay(5);

        }

    } else {

        printf("slope greater than 1");

    }

}

int main(){

    int gd = DETECT, gm;

    initgraph(&gd, &gm,(char\*) "");

    setcolor(getmaxcolor());

    drawline(50,50,450,450);

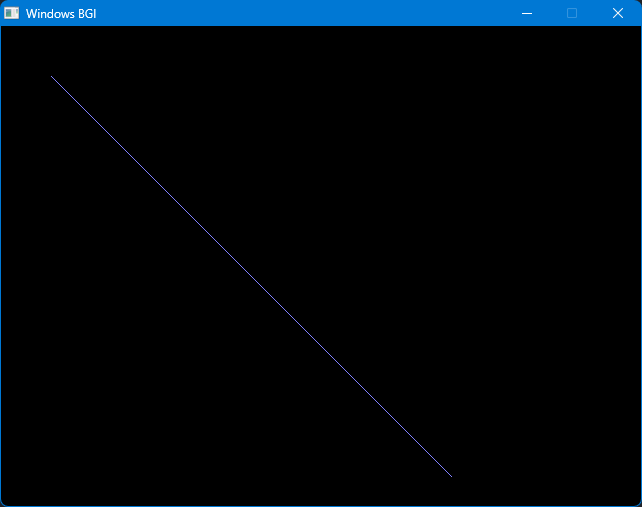
    getch();

    closegraph();

    return 0;

}

**#Output**

****

**Que2** Simulate Mid point Circle Drawing Algorithm

**Ans**

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <graphics.h>

#include <dos.h>

void **drawCircle**(int x, int y, int r)

{

    float d = 1-r;

    float col = 100;

    int x\_old = 0;

    int y\_old = r;

    while (x\_old < y\_old){

**putpixel**( x\_old+x, y\_old+y, col);

**putpixel**(-x\_old+x, y\_old+y, col);

**putpixel**( x\_old+x,-y\_old+y, col);

**putpixel**(-x\_old+x,-y\_old+y, col);

**putpixel**( y\_old+x, x\_old+y, col);

**putpixel**(-y\_old+x, x\_old+y, col);

**putpixel**( y\_old+x,-x\_old+y, col);

**putpixel**(-y\_old+x,-x\_old+y, col);

        if (d<=0){

            d = d+3+2\*x\_old;

        } else {

            d = d+5+2\*(x\_old-y\_old);

            y\_old--;

        }

        x\_old++;

**delay**(5);

        col += 0.1;

    }

}

int **main**()

{

    int gdriver = DETECT, gmode, errorcode;

**initgraph**(&gdriver, &gmode, "C:\\TURBOC3\\BGI");

    errorcode = **graphresult**();

**setcolor**(**getmaxcolor**());

**drawCircle**(200, 200, 150);

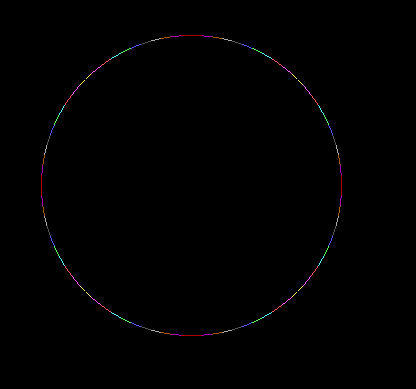
**getch**();

**closegraph**();

    return 0;

}

**#Output**



**Que3** Write a program to clip a line using Cohen and Sutherland line clipping algorithm**.**

**Ans**

#include <graphics.h>

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

using namespace std;

const int xmin = 200;

const int xmax = 400;

const int ymin = 150;

const int ymax = 350;

int get\_code(float *x*, float *y*){

    int code = 0;

    if (*x* <= xmin)

        code += 1;

    else if (*x* >= xmax)

        code += 2;

    if (*y* <= ymin)

        code += 4;

    else if (*y* >= ymax)

        code += 8;

    return code;

}

void clip\_line(float *x1*, float *y1*, float *x2*, float *y2*){

    int pt1\_code = get\_code(*x1*, *y1*);

    int pt2\_code = get\_code(*x2*, *y2*);

    if (!(pt1\_code | pt2\_code)){

        printf("Line is fully visible\n");

        setcolor(GREEN);

        line(*x1*, *y1*, *x2*, *y2*);

    }

    else{

        if (pt1\_code & pt2\_code){

            printf("Line is fully invisible\n");

            setcolor(RED);

            line(*x1*, *y1*, *x2*, *y2*);

        }

        else{

            printf("Line if partially visble\n");

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

            int x1new = *x1*;

            int y1new = *y1*;

            int x2new = *x2*;

            int y2new = *y2*;

*// \*-----------------------------------*

            if (pt1\_code & 8){

                x1new = *x1* + (ymax - *y1*) / m;

                y1new = ymax;

            }

            if (pt1\_code & 4){

                x1new = *x1* + (ymin - *y1*) / m;

                y1new = ymin;

            }

            if (pt1\_code & 2){

                y1new = *y1* + (xmax - *x1*) \* m;

                x1new = xmax;

            }

            if (pt1\_code & 1){

                y1new = *y1* + (xmin - *x1*) \* m;

                x1new = xmin;

            }

*// \*-----------------------------------*

            if (pt2\_code & 8){

                x2new = *x2* + (ymax - *y2*) / m;

                y2new = ymax;

            }

            if (pt2\_code & 4){

                x2new = *x2* + (ymin - *y2*) / m;

                y2new = ymin;

            }

            if (pt2\_code & 2){

                y2new = *y2* + (xmax - *x2*) \* m;

                x2new = xmax;

            }

            if (pt2\_code & 1){

                y2new = *y2* + (xmin - *x2*) \* m;

                x2new = xmin;

            }

*// \*-----------------------------------*

            setcolor(LIGHTGREEN);

            line(x1new,y1new,x2new,y2new);

        }

    }

}

int main(){

    int gd = DETECT, gm;

    initgraph(&gd, &gm, (char \*)"");

    float x1 = 50;

    float y1 = 120;

    float x2 = 450;

    float y2 = 320;

    setcolor(RED);

    line(x1,y1,x2,y2);

    setcolor(YELLOW);

    line(xmin,ymin,xmin,ymax);

    line(xmin,ymin,xmax,ymin);

    line(xmin,ymax,xmax,ymax);

    line(xmax,ymin,xmax,ymax);

    clip\_line(x1,y1,x2,y2);

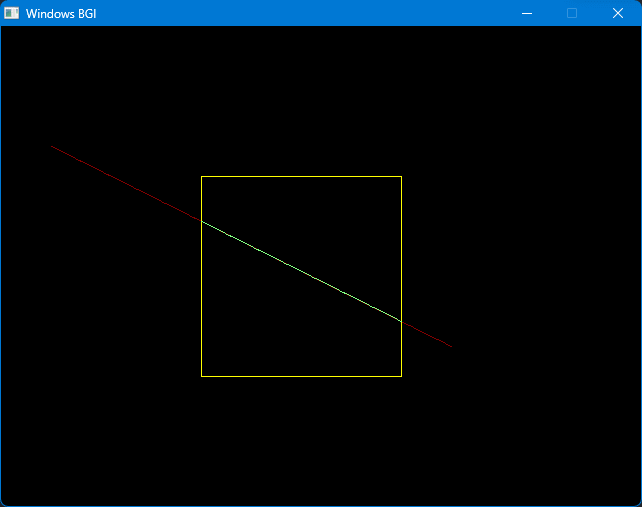
    getch();

    closegraph();

    return 0;

}

**#Output**



**Que4** Write a program to clip a polygon using Sutherland Hodgeman algorithm.

**Ans**

#include <graphics.h>

const int MAX\_POINTS = 20;

const int xmin = 200, xmax = 400, ymin = 150, ymax = 350;

void drawpoly(int poly[][2], int count, int color){

    setcolor(color);

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

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

    }

    line(poly[count-1][0], poly[count-1][1], poly[0][0], poly[0][1]);

}

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;

}

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 clip(int poly\_points[][2], int &poly\_size,int x1, int y1, int x2, int y2){

    int nP[MAX\_POINTS][2], ns = 0;

    for (int i = 0; i < poly\_size; i++){

        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];

        int i\_pos = (x2-x1) \* (iy-y1)-(y2-y1) \* (ix-x1);

        int k\_pos = (x2-x1) \* (ky-y1)-(y2-y1) \* (kx-x1);

        if (i\_pos < 0 && k\_pos < 0){

            nP[ns][0] = kx;

            nP[ns][1] = ky;

            ns++;

        } else if (i\_pos >= 0 && k\_pos < 0){

            nP[ns][0] = x\_intersect(x1, y1, x2, y2, ix, iy, kx, ky);

            nP[ns][1] = y\_intersect(x1, y1, x2, y2, ix, iy, kx, ky);

            ns++;

            nP[ns][0] = kx;

            nP[ns][1] = ky;

            ns++;

        } else if (i\_pos < 0 && k\_pos >= 0){

            nP[ns][0] = x\_intersect(x1, y1, x2, y2, ix, iy, kx, ky);

            nP[ns][1] = y\_intersect(x1, y1, x2, y2, ix, iy, kx, ky);

            ns++;

        }

    }

    poly\_size = ns;

    for (int i = 0; i < poly\_size; i++) {

        poly\_points[i][0] = nP[i][0];

        poly\_points[i][1] = nP[i][1];

    }

}

void suthHodgClip(int poly\_points[][2], int poly\_size,int ClipP[][2], int CLipS){

    for (int i = 0; i < CLipS; i++){

        int k = (i + 1) % CLipS;

        clip(poly\_points, poly\_size, ClipP[i][0],

             ClipP[i][1], ClipP[k][0],

             ClipP[k][1]);

    }

    drawpoly(poly\_points,poly\_size,LIGHTGREEN);

}

int main(){

    int gd = DETECT,gm;

    initgraph(&gd,&gm,(char \*) "");

    int poly\_size = 3;

    int poly\_points[20][2] = {{100, 150}, {200, 250}, {300, 200}};

    drawpoly(poly\_points,poly\_size,RED);

    setcolor(YELLOW);

    rectangle(xmin,ymin,xmax,ymax);

    int clipper\_size = 4;

    int clipper\_points[][2] = {{xmin, ymin}, {xmin, ymax}, {xmax, ymax}, {xmax, ymin}};

    suthHodgClip(poly\_points, poly\_size, clipper\_points, clipper\_size);

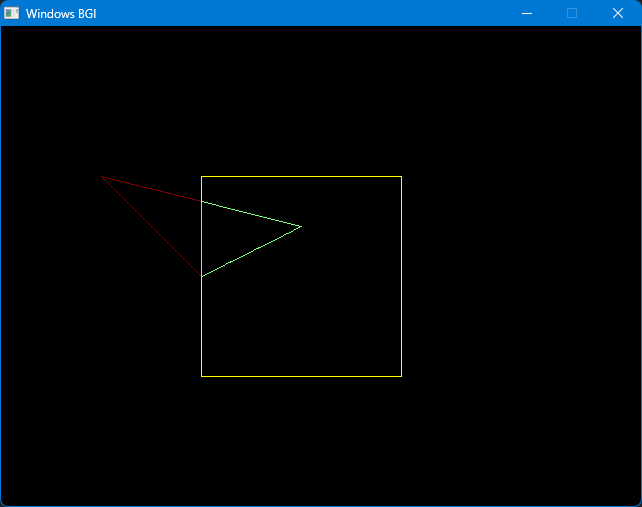
    getch();

    closegraph();

    return 0;

}

**#Output**

****

**Que5** Write a program to fill a polygon using Scan line fill algorithm.

**Ans**

#include <graphics.h>

void scanLineFill(int n, int polyPoints[]){

    int minY = INT\_MAX, maxY = INT\_MIN;

    for (int i = 1; i <= n; i += 2){

        if (polyPoints[i] < minY)

            minY = polyPoints[i];

        if (polyPoints[i] > maxY)

            maxY = polyPoints[i];

    }

    for (int y = minY; y <= maxY; y++){

        int intersectX[100], k = 0;

        for (int i = 0; i < n; i += 2){

            int x1 = polyPoints[i];

            int y1 = polyPoints[i + 1];

            int x2 = polyPoints[(i + 2) % n];

            int y2 = polyPoints[(i + 3) % n];

            if ((y >= y1 && y < y2) || (y >= y2 && y < y1)){

                if (y1 == y2) *// Horizontal line, skip*

                    continue;

                int x = x1 + (y - y1) \* (x2 - x1) / (y2 - y1);

                intersectX[k++] = x;

            }

        }

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

            for (int j = 0; j < k - i - 1; j++){

                if (intersectX[j] > intersectX[j + 1]){

                    int temp = intersectX[j];

                    intersectX[j] = intersectX[j + 1];

                    intersectX[j + 1] = temp;

                }

            }

        }

        for (int i = 0; i < k; i += 2){

            line(intersectX[i], y, intersectX[i + 1], y);

        }

    }

}

void drawpoly(int poly[][2], int count, int color){

    setcolor(color);

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

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

    }

    line(poly[count - 1][0], poly[count - 1][1], poly[0][0], poly[0][1]);

}

int main(){

    int gd = DETECT, gm;

    initgraph(&gd, &gm, (char \*)"");

    int n = 6;

    int polyPoints[] = {200, 100, 300, 200, 400, 100, 400, 300, 300, 400, 200, 300};

    int polyPoints2[][2] = {{200, 100}, {300, 200}, {400, 100}, {400, 300}, {300, 400}, {200, 300}};

    drawpoly(polyPoints2, n, RED);

    setcolor(YELLOW);

    scanLineFill(n, polyPoints);

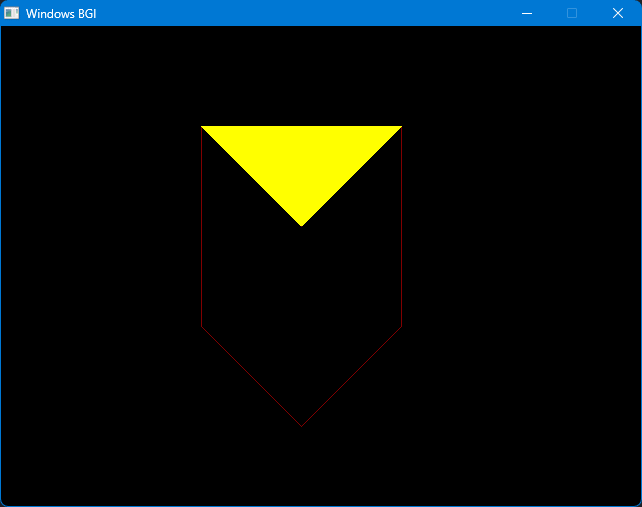
    getch();

    closegraph();

    return 0;

}

**#Output**



**Que6** Write a program to apply various 2D transformations on a 2D object (use homogenous Coordinates).

**Scaling**

**Ans**

#include <stdio.h>

#include <graphics.h>

int \*\*MatrixMul(int pcount, int P[][3], float T[3][3]){

    int\*\* res = (int\*\*)malloc(pcount \* sizeof(int\*));

    for (int i = 0; i < pcount; i++) {

        res[i] = (int\*)malloc(3 \* sizeof(int));

    }

    for (int i = 0; i < pcount; i++){

        for (int j = 0; j < 3; j++){

            res[i][j] = 0;

            for (int k = 0; k < 3; k++){

                res[i][j] += P[i][k] \* T[k][j];

            }

        }

    }

    return res;

}

int \*\*scaling(int x1,int y1,int x2,int y2,float sx,float sy){

    int points[2][3] = {{x1,y1,1},{x2,y2,1}};

    float Tmatrix[3][3] = {{sx,0,0},{0,sy,0},{0,0,1}};

    int\*\* res = MatrixMul(2,points,Tmatrix);

    return res;

}

int main(){

    int gd = DETECT, gm;

    initgraph(&gd, &gm, (char \*)"");

    int x1 = 25;

    int y1 = 30;

    int x2 = 300;

    int y2 = 110;

    float sx = 0.6;

    float sy = 2;

    int xaxis = getmaxy()/2;

    int yaxis = getmaxx()/2;

    setcolor(WHITE);

    line(yaxis,0,yaxis,getmaxy());

    line(0,xaxis,getmaxx(),xaxis);

    setcolor(GREEN);

    line(x1+yaxis,y1+xaxis,x2+yaxis,y2+xaxis);

    setcolor(LIGHTGREEN);

    int\*\* res = scaling(x1, y1, x2, y2, sx, sy);

    line(res[0][0]+yaxis,res[0][1]+xaxis,res[1][0]+yaxis,res[1][1]+xaxis);

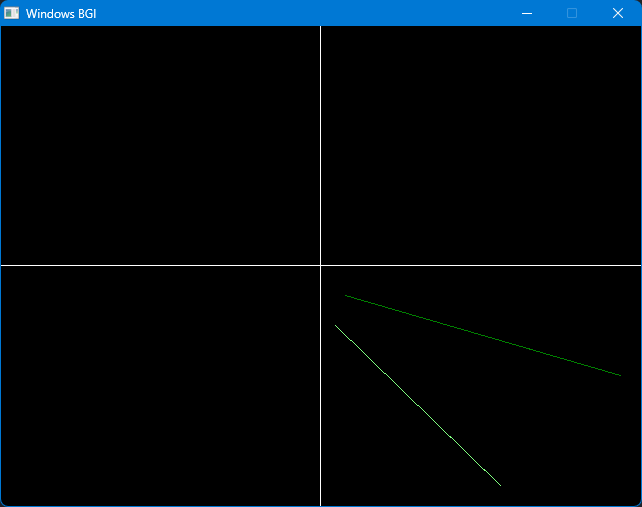
    getch();

    closegraph();

    return 0;

;}

**#Output**



**Rotation**

**Ans**

#include <stdio.h>

#include <cmath>

#include <graphics.h>

#define PI 3.1415926

int\*\* MatMul(int pcount,int P[][3],float T[3][3]){

    int\*\* prod = (int \*\*) malloc(pcount \* sizeof(int\*));

    for(int i = 0 ; i < pcount;i++){

        prod[i] = (int\* ) malloc(sizeof(int) \* 2);

    }

    for(int m = 0; m < pcount;m++){

        for(int n = 0;n < 3;n++){

            int sum = 0;

            for(int k = 0;k < 3;k++){

                sum+=P[m][k]\*T[k][n];

            }

            prod[m][n] = sum;

        }

    }

    return prod;

}

int \*\*rotation(int x1,int y1,int x2,int y2,double angle){

    angle\*=PI/180;

    int points[2][3] = {{x1,y1,1},{x2,y2,1}};

    float Tmatrix[3][3] = {{cos(angle),sin(angle),0},{-sin(angle),cos(angle),0},{0,0,1}};

    if(angle < 0){

        Tmatrix[0][1]\*=-1;Tmatrix[1][0]\*=-1;

    }

    int\*\* res = MatMul(2,points,Tmatrix);

    res[0][0] = x1;res[0][1] = y1;

    return res;

}

int main(){

int gd = DETECT, gm;

    initgraph(&gd, &gm, (char \*)"");

    int x1 = 25;

    int y1 = 30;

    int x2 = 150;

    int y2 = 110;

    double ang = 45;

    int xaxis = getmaxy()/2;

    int yaxis = getmaxx()/2;

    setcolor(WHITE);

    line(yaxis,0,yaxis,getmaxy());

    line(0,xaxis,getmaxx(),xaxis);

    setcolor(GREEN);

    line(x1+yaxis,-y1+xaxis,x2+yaxis,-y2+xaxis);

    int\*\* res = rotation(x1, y1, x2, y2, ang);

    setcolor(LIGHTGREEN);

    line(res[0][0]+yaxis,-res[0][1]+xaxis,res[1][0]+yaxis,-res[1][1]+xaxis);

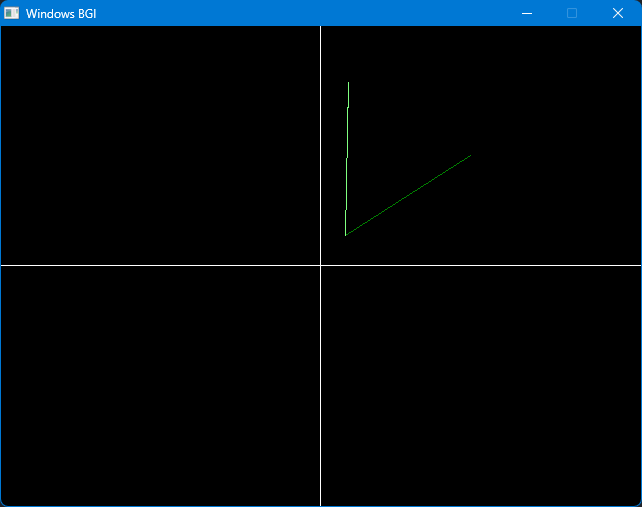
    getch();

    closegraph();

    return 0;

}

**#Output**

****

**Reflection**

#include <graphics.h>

#include <stdio.h>

#include <math.h>

int \*\*MatrixMul(int pcount, int P[][3], int T[3][3]){

    int\*\* res = (int\*\*)malloc(pcount \* sizeof(int\*));

    for (int i = 0; i < pcount; i++) {

        res[i] = (int\*)malloc(2 \* sizeof(int));

    }

    for (int i = 0; i < pcount; i++){

        for (int j = 0; j < 3; j++){

            res[i][j] = 0;

            for (int k = 0; k < 3; k++){

                res[i][j] += P[i][k] \* T[k][j];

            }

        }

    }

    return res;

}

int \*\*reflection(int x1, int y1, int x2, int y2, char axis){

    int points[][3] = {{x1,y1,1}, {x2, y2,1}};

    int\*\* a;

    if (axis == 'x'){

        int Tmatrix[3][3] = {{1,0,0},{0,-1,0},{0,0,1}};

        a = MatrixMul(2, points, Tmatrix);

    } else if (axis == 'y'){

        int Tmatrix[3][3] = {{-1,0,0},{0,1,0},{0,0,1}};

        a = MatrixMul(2, points, Tmatrix);

    }

    return a;

}

int main(){

    int gd = DETECT, gm;

    initgraph(&gd, &gm, (char \*)"");

    int x1 = 25;

    int y1 = 30;

    int x2 = 300;

    int y2 = 150;

    char axis = 'y';

    int xaxis = getmaxy()/2;

    int yaxis = getmaxx()/2;

    setcolor(WHITE);

    line(yaxis,0,yaxis,getmaxy());

    line(0,xaxis,getmaxx(),xaxis);

    setcolor(GREEN);

    line(x1+yaxis,y1+xaxis,x2+yaxis,y2+xaxis);

    setcolor(LIGHTBLUE);

    if (axis == 'x'){

        line(0,xaxis,getmaxx(),xaxis);

    } else if (axis == 'y'){

        line(yaxis,0,yaxis,getmaxy());

    }

    int\*\* res = reflection(x1, y1, x2, y2, axis);

    setcolor(LIGHTGREEN);

    line(res[0][0]+yaxis,res[0][1]+xaxis,res[1][0]+yaxis,res[1][1]+xaxis);

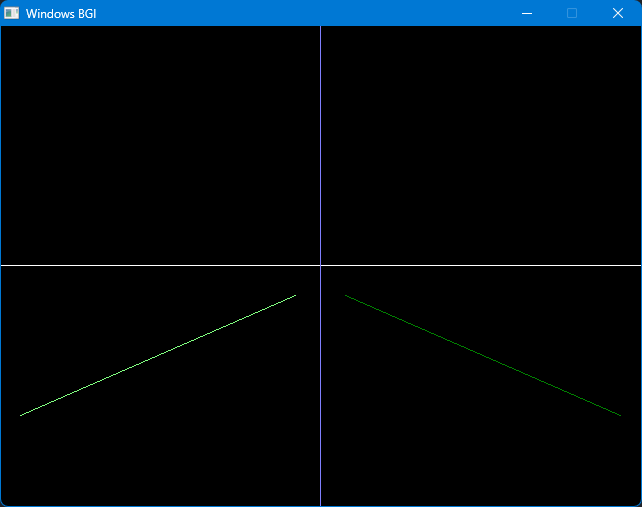
    getch();

    closegraph();

    return 0;

}

**#Output**

****

**Shearing**

#include <stdio.h>

#include <graphics.h>

int \*\*MatrixMul(int pcount, int P[][3], float T[3][3]){

    int\*\* res = (int\*\*)malloc(pcount \* sizeof(int\*));

    for (int i = 0; i < pcount; i++) {

        res[i] = (int\*)malloc(2 \* sizeof(int));

    }

    for (int i = 0; i < pcount; i++){

        for (int j = 0; j < 3; j++){

            res[i][j] = 0;

            for (int k = 0; k < 3; k++){

                res[i][j] += P[i][k] \* T[k][j];

            }

        }

    }

    return res;

}

int \*\*shearing(int x1,int y1,int x2,int y2,float sx,float sy){

    int points[2][3] = {{x1,y1,1},{x2,y2,1}};

    float Tmatrix[3][3] = {{1+(sx\*sy),sx,0},{sy,1+(sx\*sy),0},{0,0,1}};

    int\*\* res = MatrixMul(2,points,Tmatrix);

    return res;

}

int main(){

    int gd = DETECT, gm;

    initgraph(&gd, &gm, (char \*)"");

    int x1 = 25;

    int y1 = 30;

    int x2 = 300;

    int y2 = 110;

    float sx = 1;

    float sy = 1.2;

    int xaxis = getmaxy()/2;

    int yaxis = getmaxx()/2;

    setcolor(WHITE);

    line(yaxis,0,yaxis,getmaxy());

    line(0,xaxis,getmaxx(),xaxis);

    setcolor(GREEN);

    line(x1+yaxis,y1+xaxis,x2+yaxis,y2+xaxis);

    int\*\* res = shearing(x1, y1, x2, y2, sx, sy);

    setcolor(LIGHTGREEN);

    line(res[0][0]+yaxis,res[0][1]+xaxis,res[1][0]+yaxis,res[1][1]+xaxis);

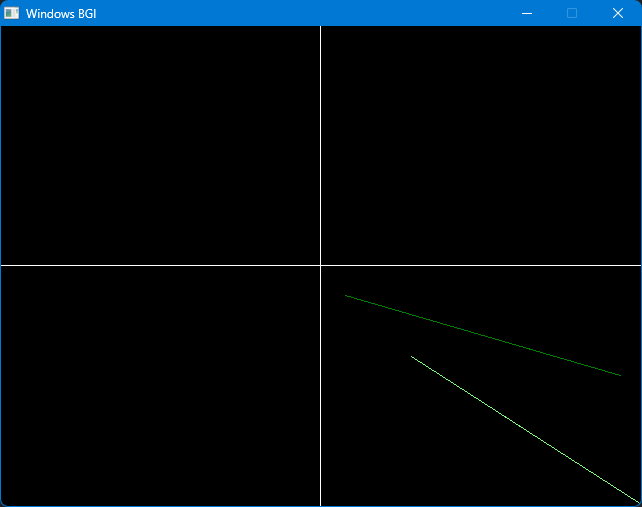
    getch();

    closegraph();

    return 0;

}

**#Output**

****

**Translation**

**Ans**

#include <graphics.h>

#include <stdio.h>

int \*\*MatrixMul(int pcount, int P[][3], float T[3][3]){

    int\*\* res = (int\*\*)malloc(pcount \* sizeof(int\*));

    for (int i = 0; i < pcount; i++) {

        res[i] = (int\*)malloc(2 \* sizeof(int));

    }

    for (int i = 0; i < pcount; i++){

        for (int j = 0; j < 3; j++){

            res[i][j] = 0;

            for (int k = 0; k < 3; k++){

                res[i][j] += P[i][k] \* T[k][j];

            }

        }

    }

    return res;

}

int \*\*translate(int x1,int y1,int x2,int y2,float tx,float ty){

    int points[2][3] = {{x1,y1,1},{x2,y2,1}};

    float Tmatrix[3][3] = {{1,0,0},{0,1,0},{tx,ty,1}};

    int\*\* res = MatrixMul(2,points,Tmatrix);

    return res;

}

int main(){

    int gd = DETECT, gm;

    initgraph(&gd, &gm, (char \*)"");

    int x1 = 25;

    int y1 = 30;

    int x2 = 300;

    int y2 = 110;

    float tx = -40;

    float ty = -120;

    int xaxis = getmaxy()/2;

    int yaxis = getmaxx()/2;

    setcolor(WHITE);

    line(yaxis,0,yaxis,getmaxy());

    line(0,xaxis,getmaxx(),xaxis);

    setcolor(GREEN);

    line(x1+yaxis,y1+xaxis,x2+yaxis,y2+xaxis);

    int\*\* res = translate(x1, y1, x2, y2, tx, ty);

    setcolor(LIGHTGREEN);

    line(res[0][0]+yaxis,res[0][1]+xaxis,res[1][0]+yaxis,res[1][1]+xaxis);

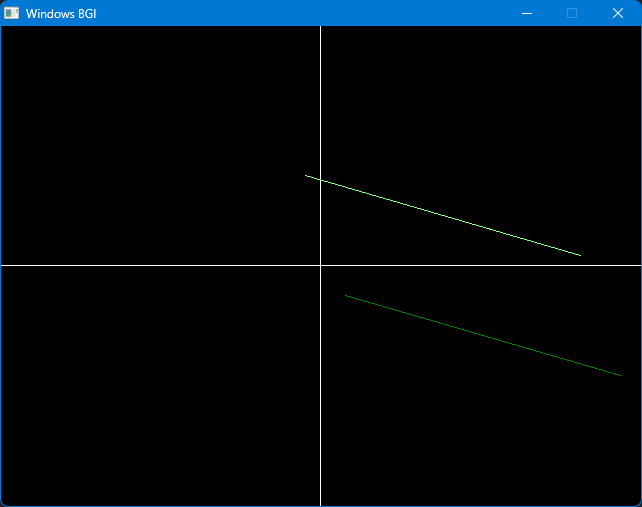
    getch();

    closegraph();

    return 0;

}

**#Output**

****

**Que7** Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it.

**Ans**

#include <graphics.h>

#include <iostream>

#include <conio.h>

#include <math.h>

using namespace std;

int gd, gm, i, ch1, ch2,y1;

double x1, x2, y2, x, y, z, theta, temp, temp1;

void draw(double[20][3]);

int main(){

    do

    {

        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, 100, 100,

                              100, 100, 100,

                              100, 100, 100,

                              100, 100, 0,

                              100, 100, 100,

                              100, 100, 100,

                              100, 100, 100,

                              0, 100, 100,

                              0, 100, 0,

                              0, 0, 0,

                              0, 0, 100,

                              100, 0, 100};

        cleardevice();

        cout << "Choose any one 3D Transformation : ";

        cout << "\n\t1. Translation ";

        cout << "\n\t2. Rotation ";

        cout << "\n\t3. Scaling ";

        cout << "\nEnter your choice : \t";

        cin >> ch1;

        switch (ch1)

        {

        case 1:

            cout << "\nEnter the translation factors(tx,ty,tz) : \t";

            cin >> x >> y >> z;

            draw(edge);

            for (i = 0; i < 20; i++)

            {

                edge[i][0] += x;

                edge[i][1] += y;

                edge[i][2] += z;

            }

            draw(edge);

            break;

        case 2:

            cout << "\n\n\t1.Rotation about X Axis ";

            cout << "\n\t2. Rotation about Y Axis ";

            cout << "\n\t3. Rotation about Z Axis ";

            cout << "\nEnter your choice : \t";

            cin >> ch2;

            cout << "\n\nEnter the angle of rotation : \t";

            cin >> theta;

            theta = (theta \* 3.14) / 180;

            switch (ch2)

            {

            case 1:

                draw(edge);

                for (i = 0; i < 20; i++){

                    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(edge);

                break;

            case 2:

                draw(edge);

                for (i = 0; i < 20; i++){

                    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(edge);

                break;

            case 3:

                draw(edge);

                for (i = 0; i < 20; i++){

                    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(edge);

                break;

            }

            break;

        case 3:

            cout << "\n\nEnter the scaling factors (sx,sy,sz) : \t";

            cin >> x >> y >> z;

            draw(edge);

            for (i = 0; i < 20; i++){

                edge[i][0] \*= x;

                edge[i][1] \*= y;

                edge[i][2] \*= z;

            }

            draw(edge);

            break;

        }

    } while (ch1 < 4);

    return 0;

}

void draw(double edge[20][3]){

    initgraph(&gd, &gm,(char \*) "");

    line(320, 240, 320, 25);

    line(320, 240, 550, 240);

    line(320, 240, 150, 410);

    for (int 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);

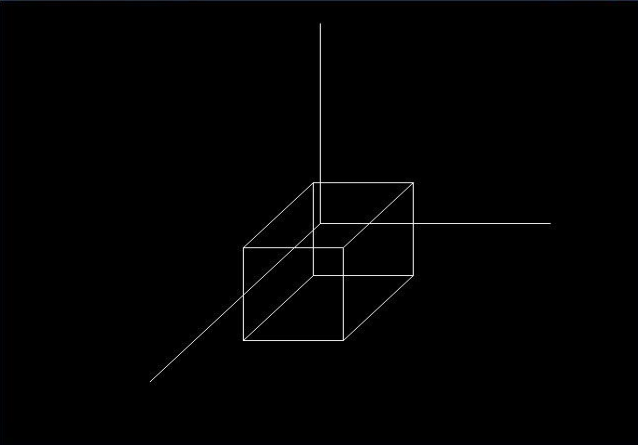
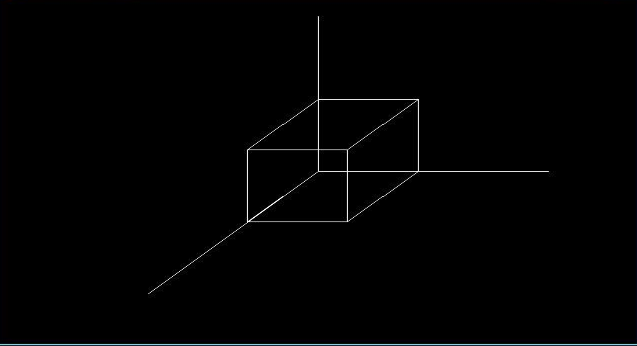
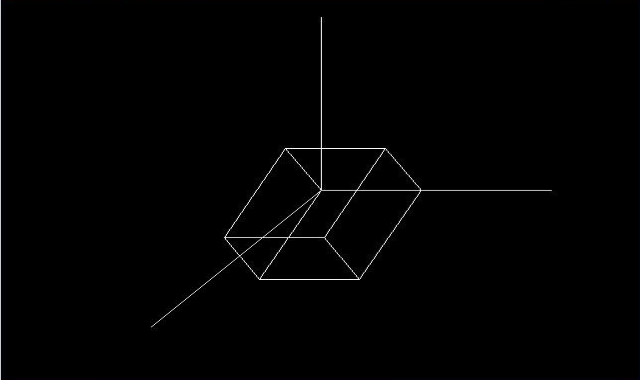
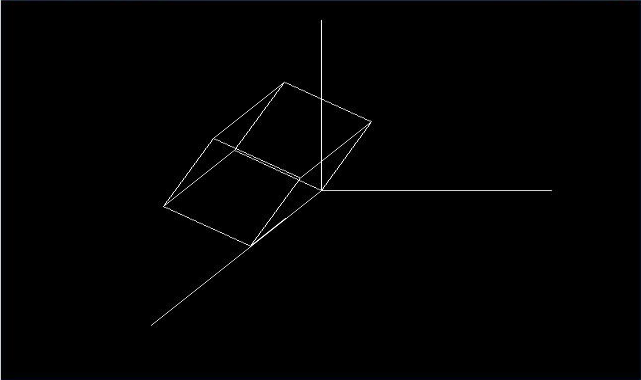
    }

    getch();

    closegraph();

}

**#Output**

**** ****  

**Que8** Write a program to draw Hermite /Bezier curve.

**Ans**

#include <graphics.h>

#include <iostream>

using namespace std;

float HB(int i, float t){

    if (i == 0)

        return (2 \* t \* t \* t - 3 \* t \* t + 1);

    else if (i == 1)

        return (-2 \* t \* t \* t + 3 \* t \* t);

    else if (i == 2)

        return (t \* t \* t - 2 \* t \* t + t);

    else

        return (t \* t \* t - t \* t);

}

float BB(int i, int n, float t){

    if (n == 0){

        if (i == 0)

            return 1;

        else

            return 0;

    }

    return (1 - t) \* BB(i, n - 1, t) + t \* BB(i - 1, n - 1, t);

}

void drawHermite(int x0, int y0, int x1, int y1, int rx0, int ry0, int rx1, int ry1){

    float step = 0.01;

    for (float t = 0; t <= 1; t += step){

        float x = x0 \* HB(0, t) + x1 \* HB(1, t) + rx0 \* HB(2, t) + rx1 \* HB(3, t);

        float y = y0 \* HB(0, t) + y1 \* HB(1, t) + ry0 \* HB(2, t) + ry1 \* HB(3, t);

        putpixel(x, y, LIGHTGREEN);

    }

}

void drawBezier(int x[], int y[], int n){

    float step = 0.01;

    for (float t = 0; t <= 1; t += step){

        float px = 0, py = 0;

        for (int i = 0; i <= n; i++){

            px += x[i] \* BB(i, n, t);

            py += y[i] \* BB(i, n, t);

        }

        putpixel(px, py, LIGHTBLUE);

    }

}

int main(){

    int gd = DETECT, gm;

    initgraph(&gd, &gm,(char \*) "");

    int x0 = 100, y0 = 250, x1 = 400, y1 = 200;

    int rx0 = 100, ry0 = 200, rx1 = 200, ry1 = 100;

    int xBezier[] = {100, 300, 200};

    int yBezier[] = {200, 400, 100};

    int n = sizeof(xBezier) / sizeof(xBezier[0]) - 1;

    drawHermite(x0, y0, x1, y1, rx0, ry0, rx1, ry1);

    drawBezier(xBezier, yBezier, n);

getch();

    closegraph();

    return 0;

}

**#Output**

