**COMPUTER GRAPHICS**

**PRACTICAL FILE**

**Semester-6th**



**Name: Pratiksha Gaur**

**Roll no.:2019/1429**

**University Roll no.: 19020570027**

**Course: B. Sc. (Hons) Computer Science**

|  |  |
| --- | --- |
| S.No. | CONTENT |
| 1 | **DDA ALGORITHM** |
| 2 | **BRESENHAM’S LINE DRAWING ALGORITHM** |
| 3 | **MID POINT CIRCLE DRAWING ALGORITHM** |
| 4 | **MID POINT ELLIPSE DRAWING ALGORITHM** |
| 5 | **COHEN SUTHERLAND LINE CLIPPING ALGORITHM** |
| 6 | **SUTHERLAND HODGEMAN POLYGON CLIPPING ALGORITHM** |
| 7 | **POLYGON SCANLINE FILLING ALGORITHM** |
| 8 | **APPLYING 2D TRANSFORMATIONS 2D OBJECT** |
| 9 | **APPLYING 3D TRANSFORMATIONS 3D OBJECT** |
| 10 | **HERMITE AND BEZIER CURVE** |

**Q1. Write a program to implement Digital Differential Analyzer line drawing algorithm.**

#include<iostream>

#include<graphics.h>

#include<windows.h>

using namespace std;

int xmid,ymid;

//Function to implement DDA line drawing algorithm

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

{

int dx,dy,steps,xinc,yinc;

dx=x2-x1;

dy=y2-y1;

xmid=getmaxx()/2;

ymid=getmaxy()/2;

if(abs (dx) > abs(dy) )

{

steps =abs(dx);

}

else

{

steps=abs(dy);

}

xinc = dx/(float) steps;

yinc = dy/(float)steps;

for(int k=0;k<steps; k++)

{

putpixel(x1,y1,YELLOW);

x1+= xinc;

y1+= yinc;

}

}

int main()

{

int gd = DETECT , gm;

initgraph(&gd, &gm,"C:\\Dev-Cpp\\lib");

int x1,y1,x2,y2;

cout<<" Digital Differential Analyzer 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;

xmid=getmaxx()/2;

ymid=getmaxy()/2;

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

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

dda(x1+xmid ,ymid-y1,x2+xmid,ymid-y2);

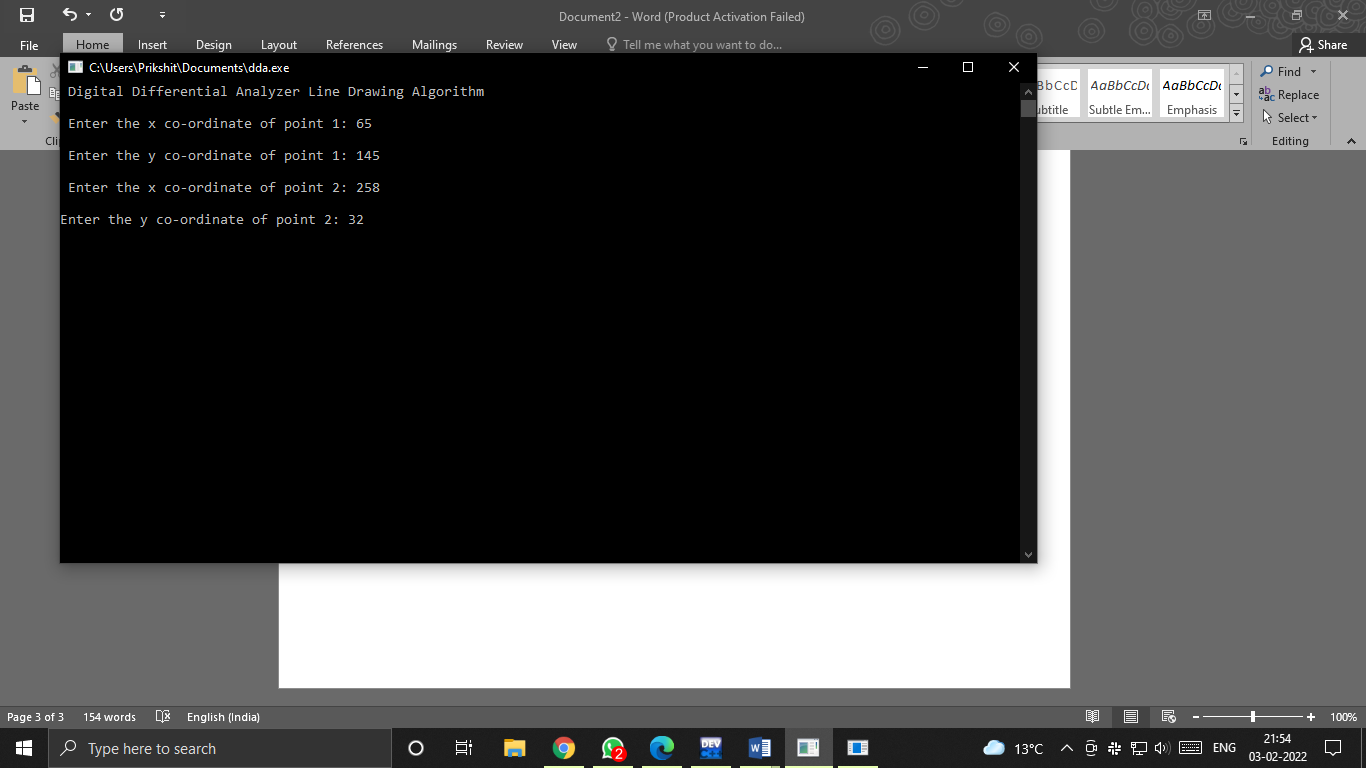
getch();

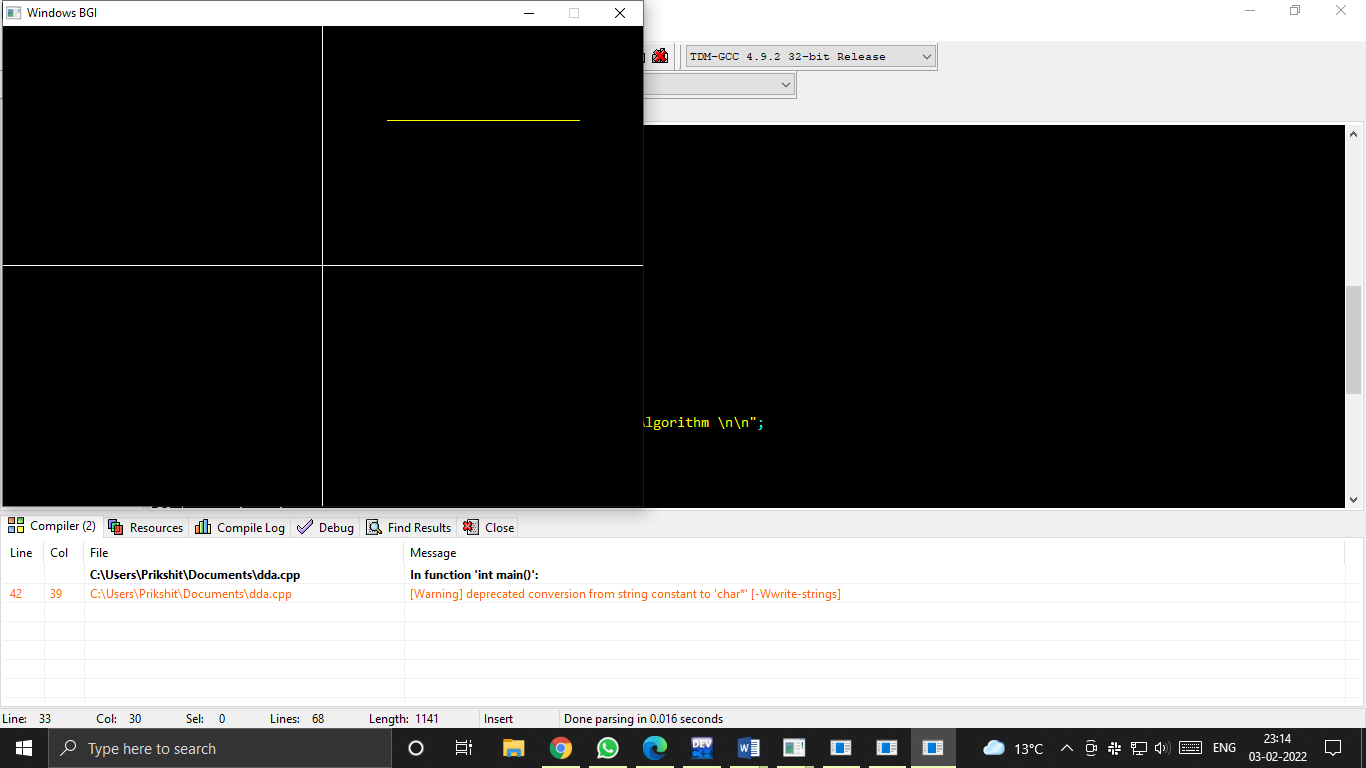
closegraph();

return 0;

}

**OUTPUT**





**Q2. Write a program to implement Bresenham’s line drawing algorithm.**

#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);

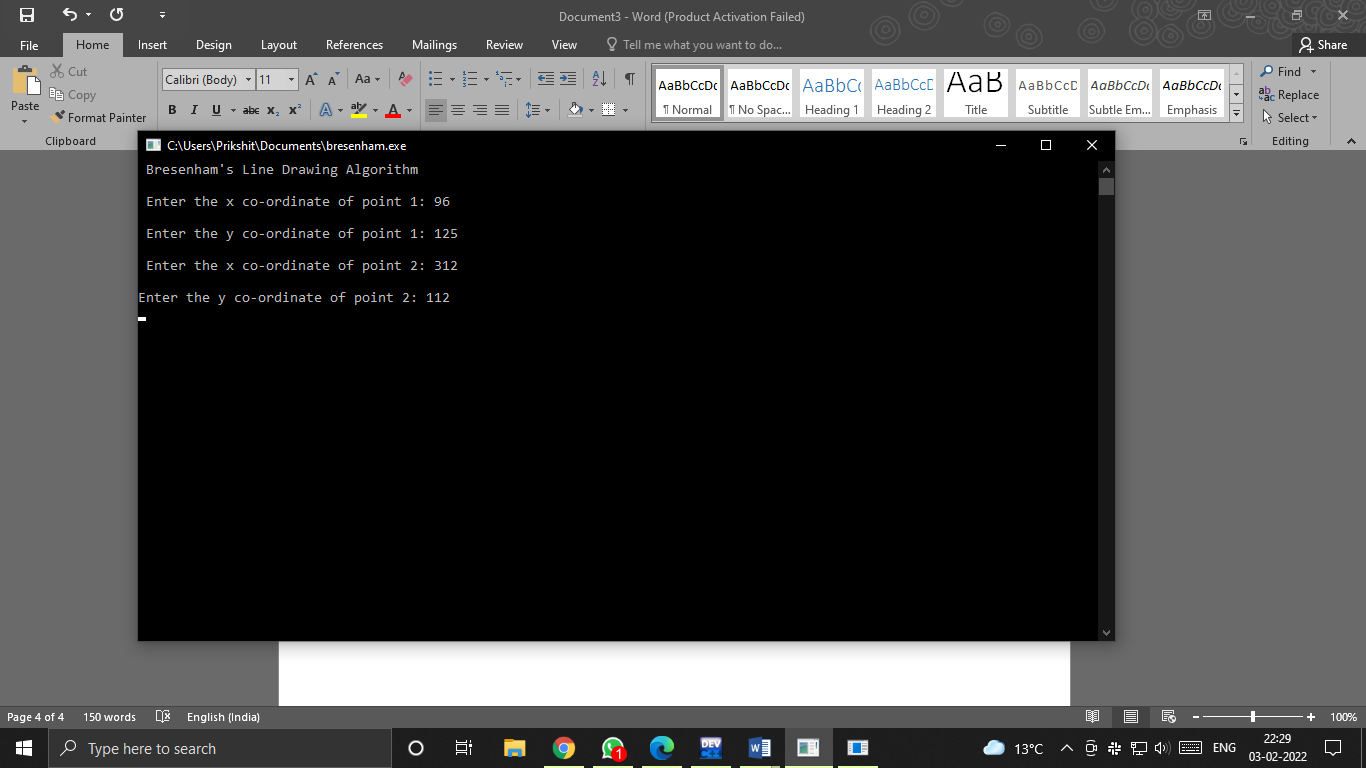
getch();

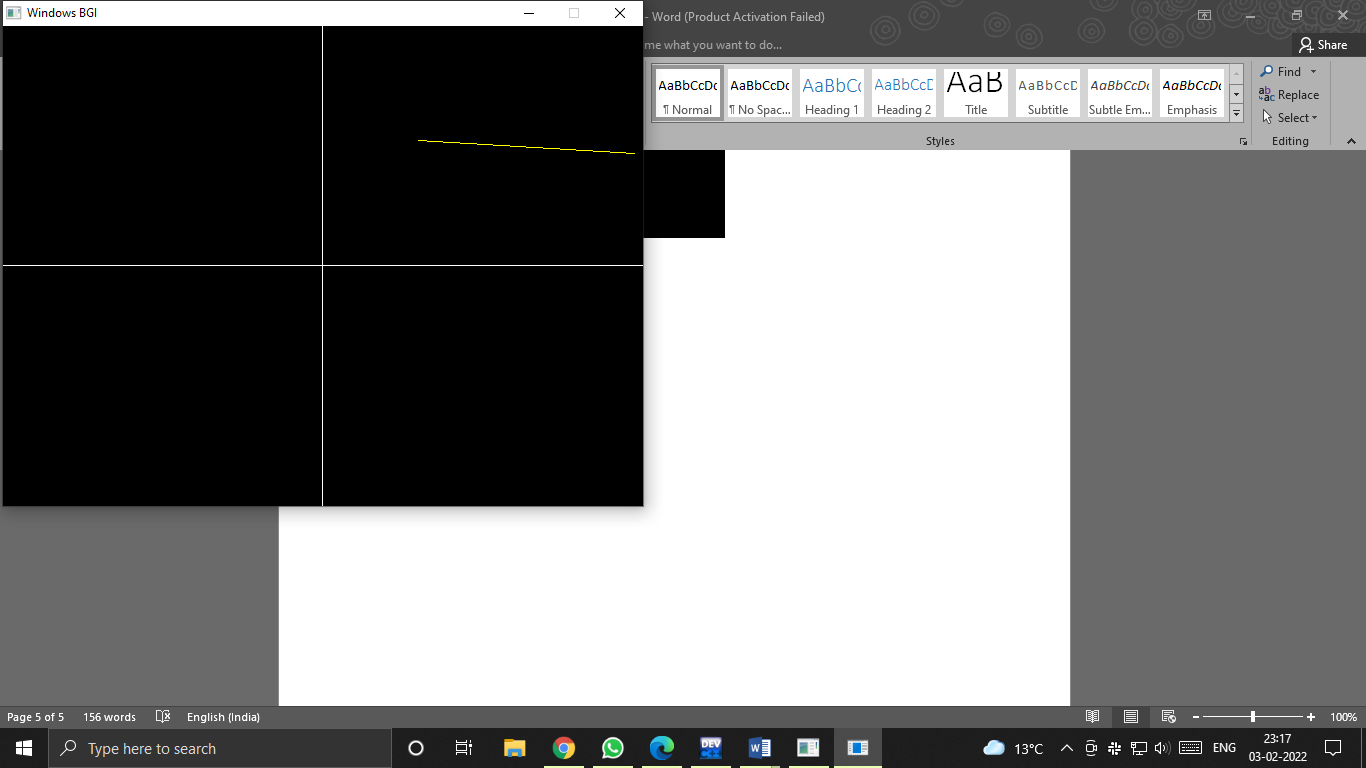
closegraph();

return 0;

}

**OUTPUT**





**Q3. Write a program to implement mid-point circle drawing algorithm.**

#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);

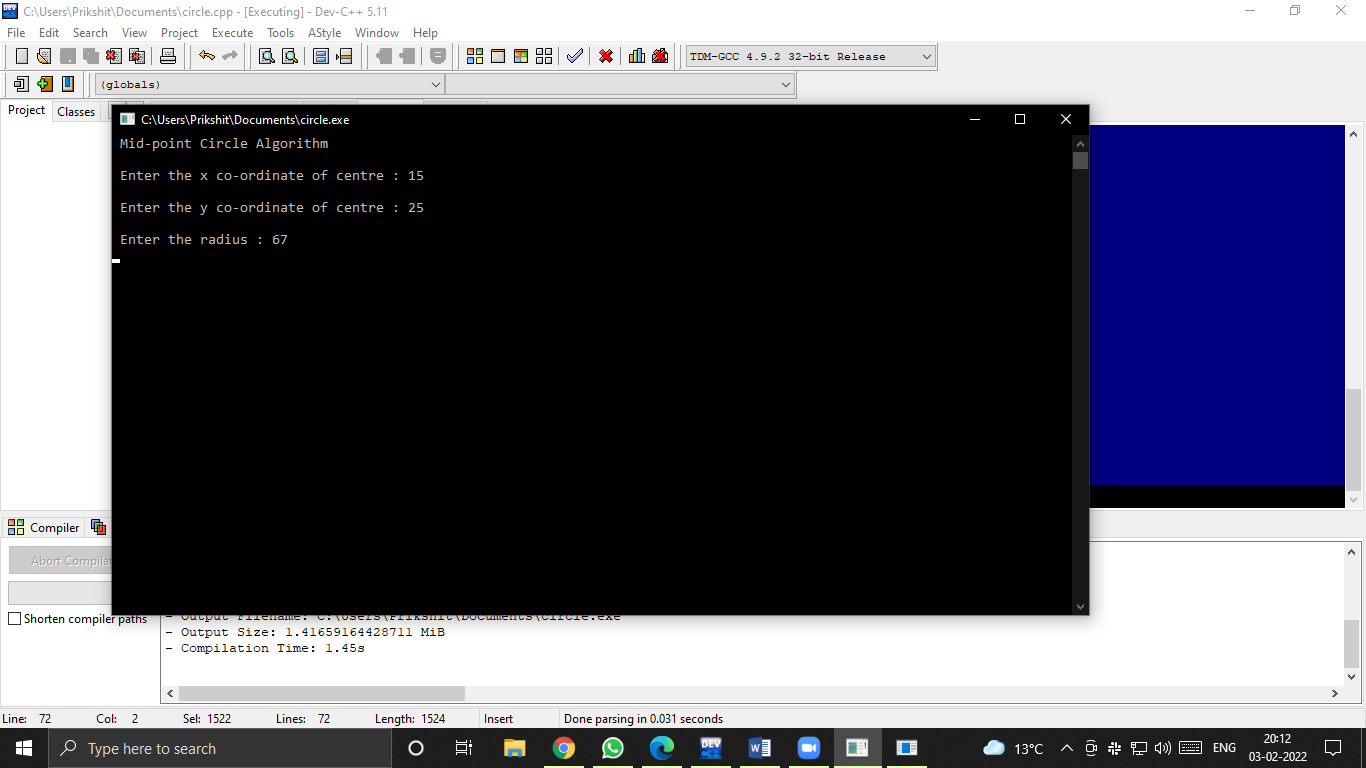
getch();

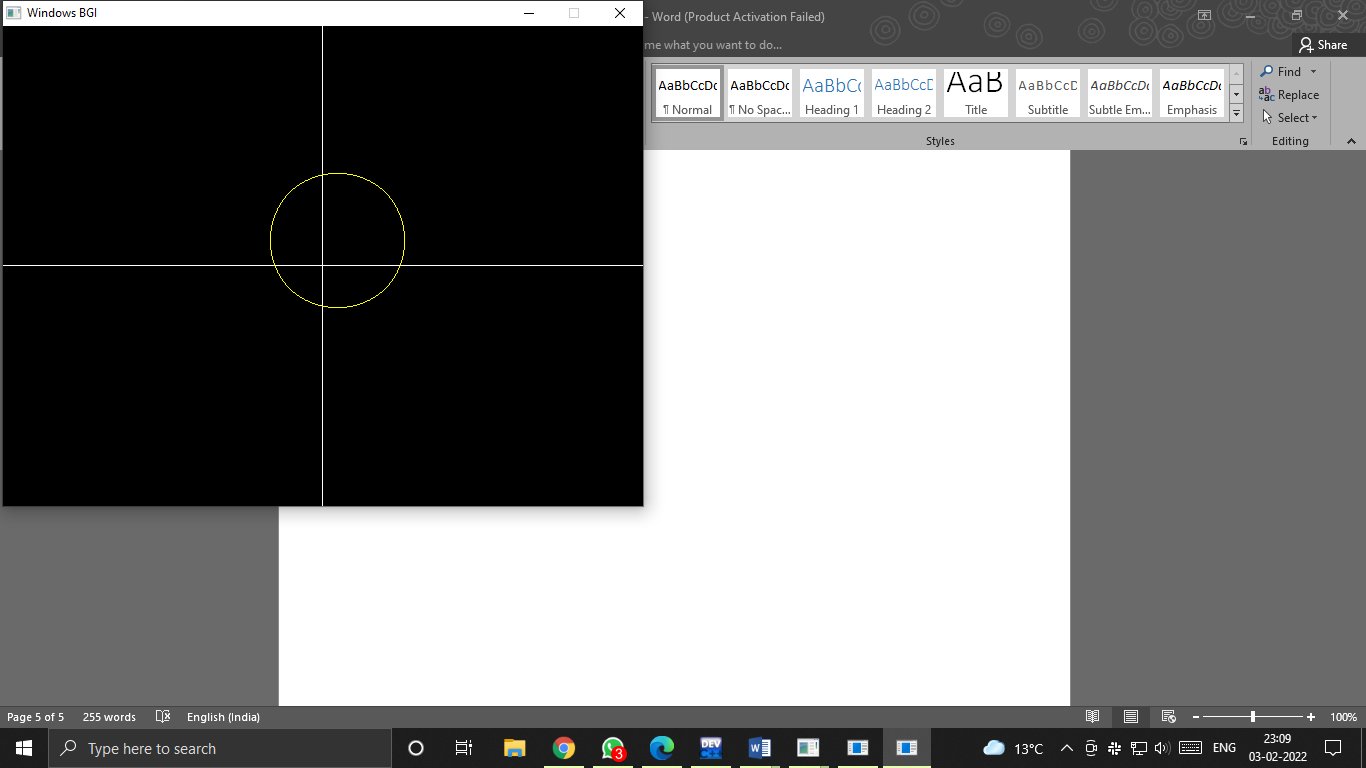
closegraph();

return 0;

}

**OUTPUT**





**Q4. Write a program to implement Ellipse mid-point drawing algorithm.**

#include<iostream>

#include<graphics.h>

#include<math.h>

using namespace std;

#define ROUND(a) ((int) (a+0.5))

void ellipsePlotPoints(int, int, int, int);

//Function plotting points of Ellipse

void ellipseMidpoint (int xCenter, int yCenter, int Rx, int Ry)

{

int Rx2 = Rx\*Rx;

int Ry2 = Ry\*Ry;

int twoRx2 = 2\*Rx2;

int twoRy2 = 2\*Ry2;

int p;

int x = 0;

int y = Ry;

int px = 0;

int py = twoRx2 \*y;

ellipsePlotPoints(xCenter, yCenter, x, y);

p = ROUND(Ry2 - (Rx2 \* Ry) + (0.25 \* Rx2));

while (px < py)

{

x++;

px += twoRy2;

if (p < 0)

{

p += Ry2 + px;

}

else

{

y--;

py -= twoRx2;

p += Ry2 + px - py;

}

ellipsePlotPoints(xCenter, yCenter, x,y);

}

/\* Region 2 \*/

p = ROUND (Ry2\*(x+0.5)\*(x+0.5) + Rx2\*(y-1)\*(y-1) - Rx2\*Ry2);

while (y > 0)

{

y--;

py -= twoRx2;

if (p > 0)

{

p += Rx2 - py;

}

else

{

x++;

px += twoRy2;

p += Rx2 - py + px;

}

ellipsePlotPoints(xCenter, yCenter, x, y);

}

}

void ellipsePlotPoints (int xCenter, int yCenter, int x, int y)

{

putpixel (xCenter + x, yCenter + y, YELLOW);

putpixel (xCenter- x, yCenter + y, YELLOW);

putpixel (xCenter+ x, yCenter - y, YELLOW);

putpixel (xCenter - x, yCenter - y, YELLOW);

}

int main()

{

int x , y,xmid,ymid;

float r,r2;

int gd = DETECT , gm;

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

cout<<" Ellipse Mid-point 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 radius1 : ";

cin>>r;

cout<<"\n Enter the radius2 : ";

cin>>r2;

xmid = getmaxx()/2;

ymid = getmaxy()/2;

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

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

ellipseMidpoint(x + xmid , ymid - y , r,r2);

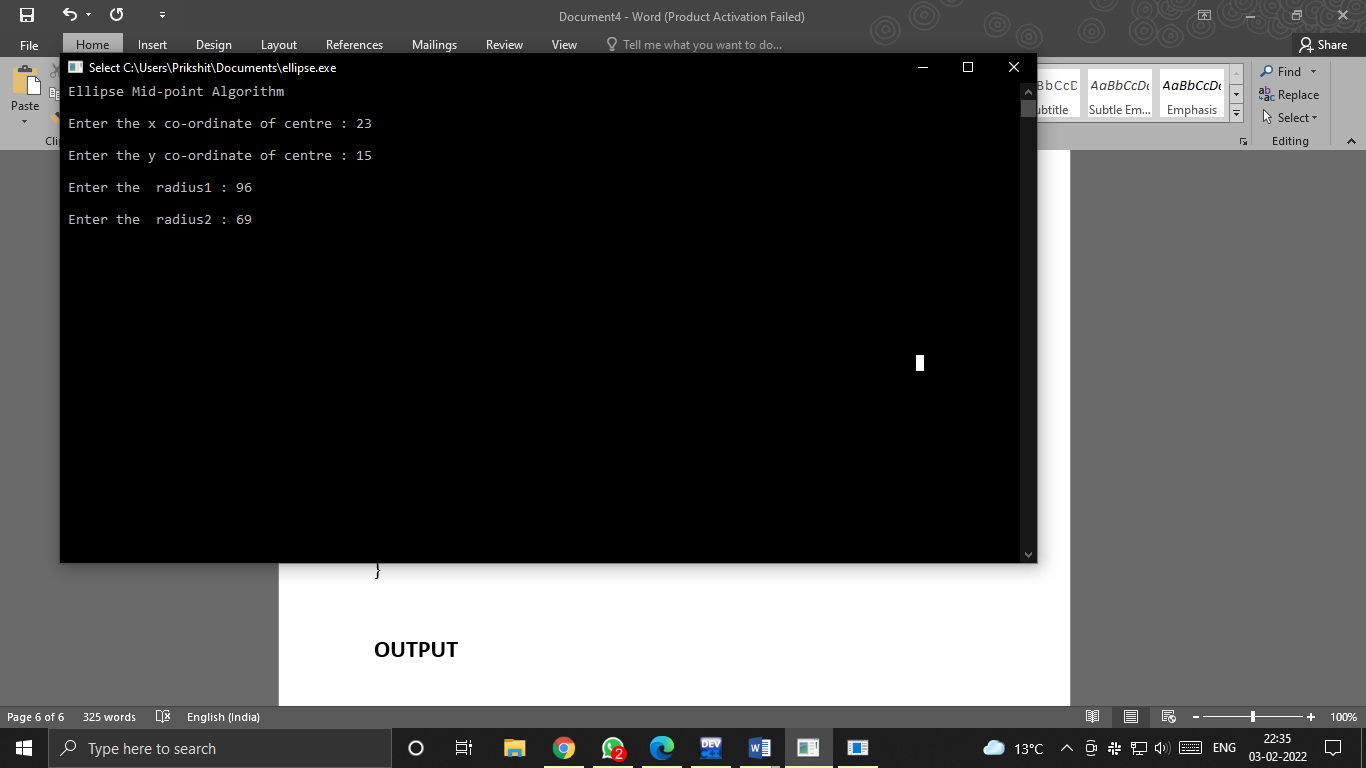
getch();

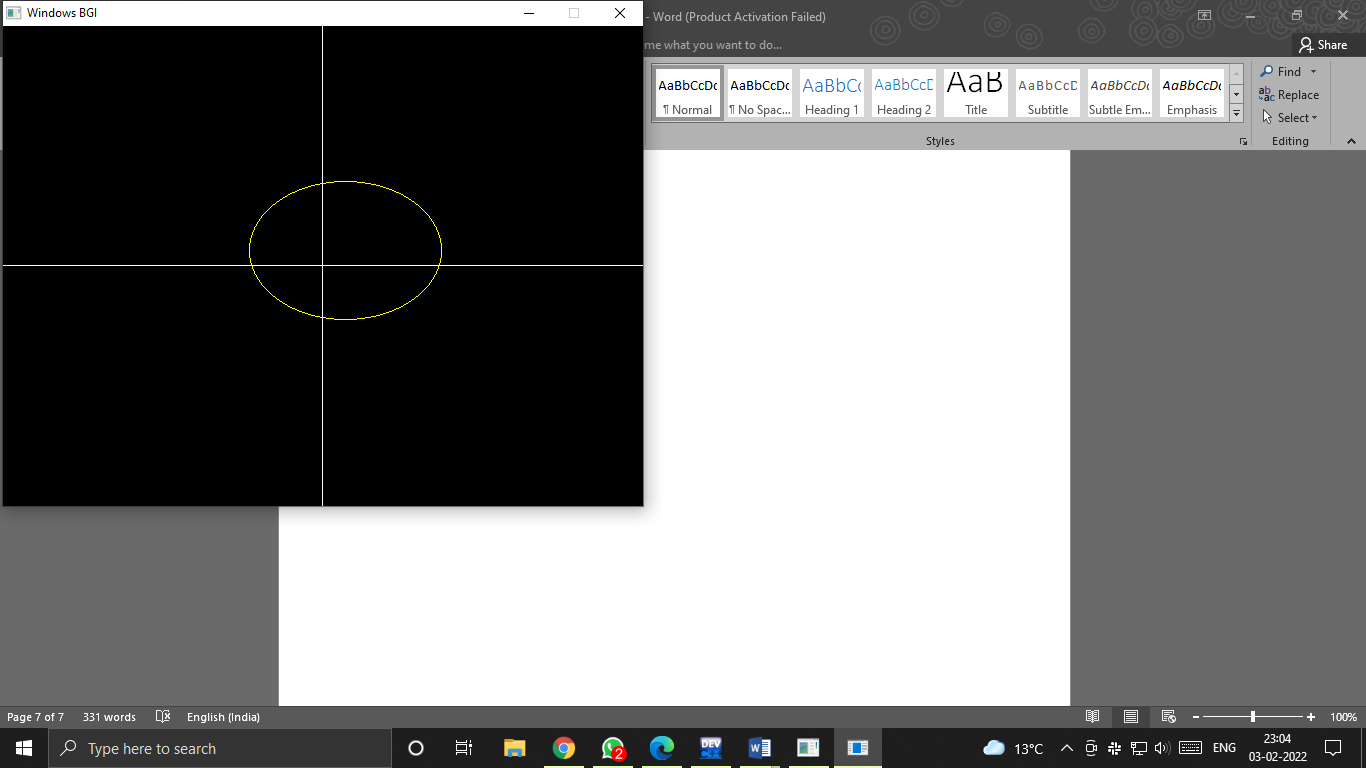
closegraph();

return 0;

}

**OUTPUT**





**Q5. Write a program to implement Cohen-Sutherland Line Clipping algorithm.**

#include <iostream>

#include<graphics.h>

#include<math.h>

using namespace std;

float x\_mid, y\_mid;

// Defining region codes

const int TOP = 1; // 0001

const int BOTTOM = 2; // 0010

const int RIGHT = 4; // 0100

const int LEFT = 8; // 1000

// Defining x\_max, y\_max and x\_min, y\_min for clipping rectangle.

const int x\_max = 300;

const int y\_max = 300;

const int x\_min = 80;

const int y\_min = 80;

// Function to compute region code for a point(x, y).

int ComputeOutCode(double x, double y)

{

// Point initialized as being inside the clipping window.

int code = 0;

if (y > y\_max)

code |= TOP;

else if (y < y\_min)

code |= BOTTOM;

if (x > x\_max)

code |= RIGHT;

else if (x < x\_min)

code |= LEFT;

return code;

}

// Implementing Cohen-Sutherland algorithm.

void CohenSutherlandLineClipAndDraw(double x1, double y1, double x2, double y2)

{

// Initialize line as outside the clipping window.

bool accept = false, done = false;

// Compute region codes for P1, P2.

int code1 = ComputeOutCode(x1, y1);

int code2 = ComputeOutCode(x2, y2);

do

{

if (!(code1 | code2))

{

// Trivial accept and exit.

accept = true;

done = true;

break;

}

else if (code1 & code2)

{

// If both endpoints are outside clipping window, so trivial reject.

break;

}

else

{

/\* Failed both tests, so calculate the line segment to clip:

from an outside point to an intersection with clip edge.

\*/

double x, y;

int code\_out;

// At least one endpoint is outside the clip rectangle, pick it.

code\_out =(code1 != 0)? code1 : code2;

// Now, find intersection point.

// Using formulas: y = y1 + slope \* (x - x1), x = x1 + (1 / slope) \* (y - y1).

if (code\_out & TOP)

{

// Point is above the clipping window.

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

y = y\_max;

}

else if (code\_out & BOTTOM)

{

// Point is below the clipping window.

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

y = y\_min;

}

else if (code\_out & RIGHT)

{

// Point is to the right of clipping window.

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

x = x\_max;

}

else if (code\_out & LEFT)

{

// Point is to the left of clipping window.

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

x = x\_min;

}

// Now we move outside point to intersection point to clip.

if (code\_out == code1)

{

x1 = x;

y1 = y;

code1 = ComputeOutCode(x1, y1);

}

else

{

x2 = x;

y2 = y;

code2 = ComputeOutCode(x2, y2);

}

}

} while(done == false);

if (accept)

{

// Drawing the clipped line.

cout << "Line accepted from (" << x1 << ", " << y1 << ") to (" << x2 << ", " << y2 << ")" << endl;

setcolor(RED);

line(x1, y1, x2, y2);

}

else

cout << "Line rejected" << endl;

}

// Driver code

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;

setcolor(WHITE);

outtextxy(30, 30, "Cohen-Sutherland Line Clipping Algorithm");

// Drawing Window using Lines

setcolor(YELLOW);

line(x\_min, y\_min, x\_max, y\_min);

line(x\_max, y\_min, x\_max, y\_max);

line(x\_max, y\_max, x\_min, y\_max);

line(x\_min, y\_max, x\_min, y\_min);

setcolor(GREEN);

// First Line segment

// P1 = (250, 320), P2 = (330, 270)

line(250, 320, 330, 270);

CohenSutherlandLineClipAndDraw(250, 320, 330, 270);

// Second Line segment

// P1 = (80, 80), P2 = (150, 150)

CohenSutherlandLineClipAndDraw(100, 100, 150, 150);

// Third Line segment

// P1 = (290, 310), P2 = (320, 500)

setcolor(GREEN);

line(290, 310, 300, 400);

CohenSutherlandLineClipAndDraw(290, 310, 320, 400);

// Fourth Line segment

// P1 = (450, 450), P2 = (500, 500)

setcolor(GREEN);

line(350, 150, 450, 250);

CohenSutherlandLineClipAndDraw(350, 150, 450, 250);

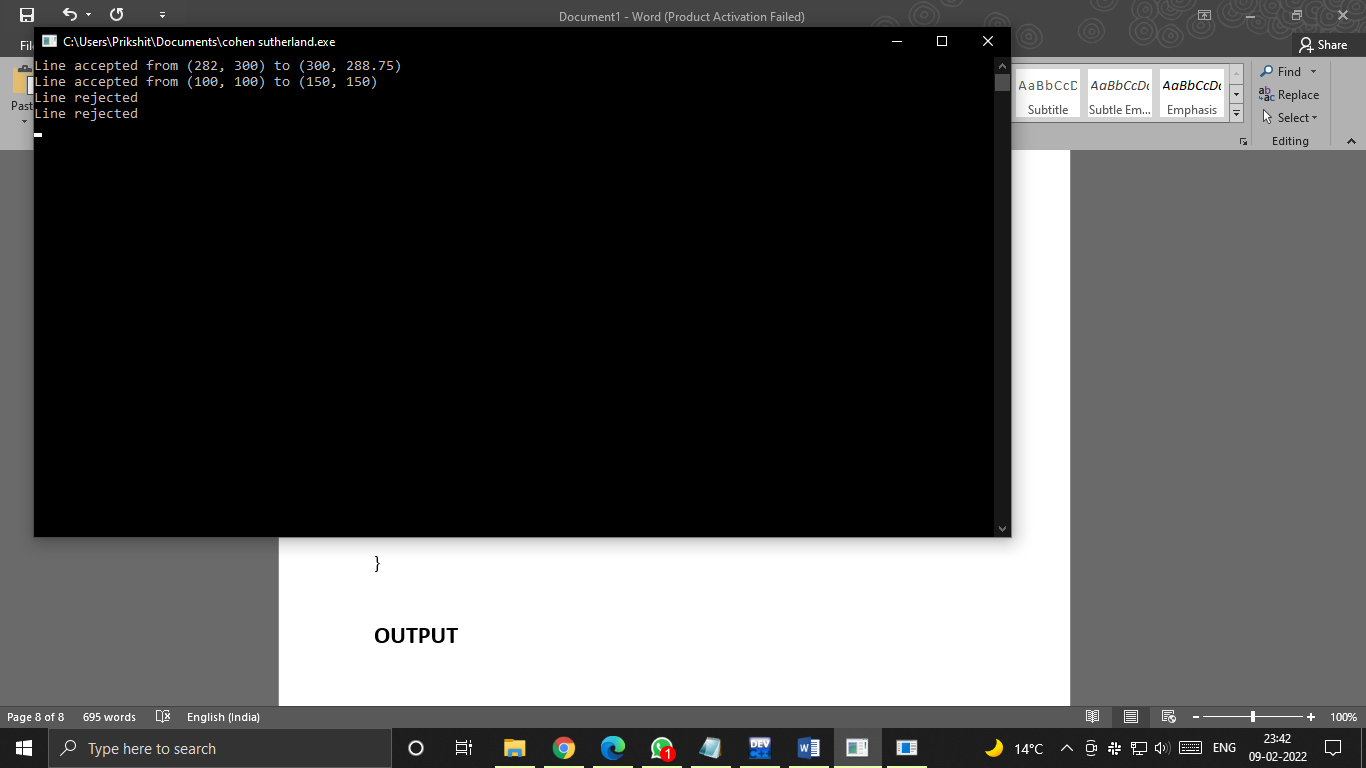
getch();

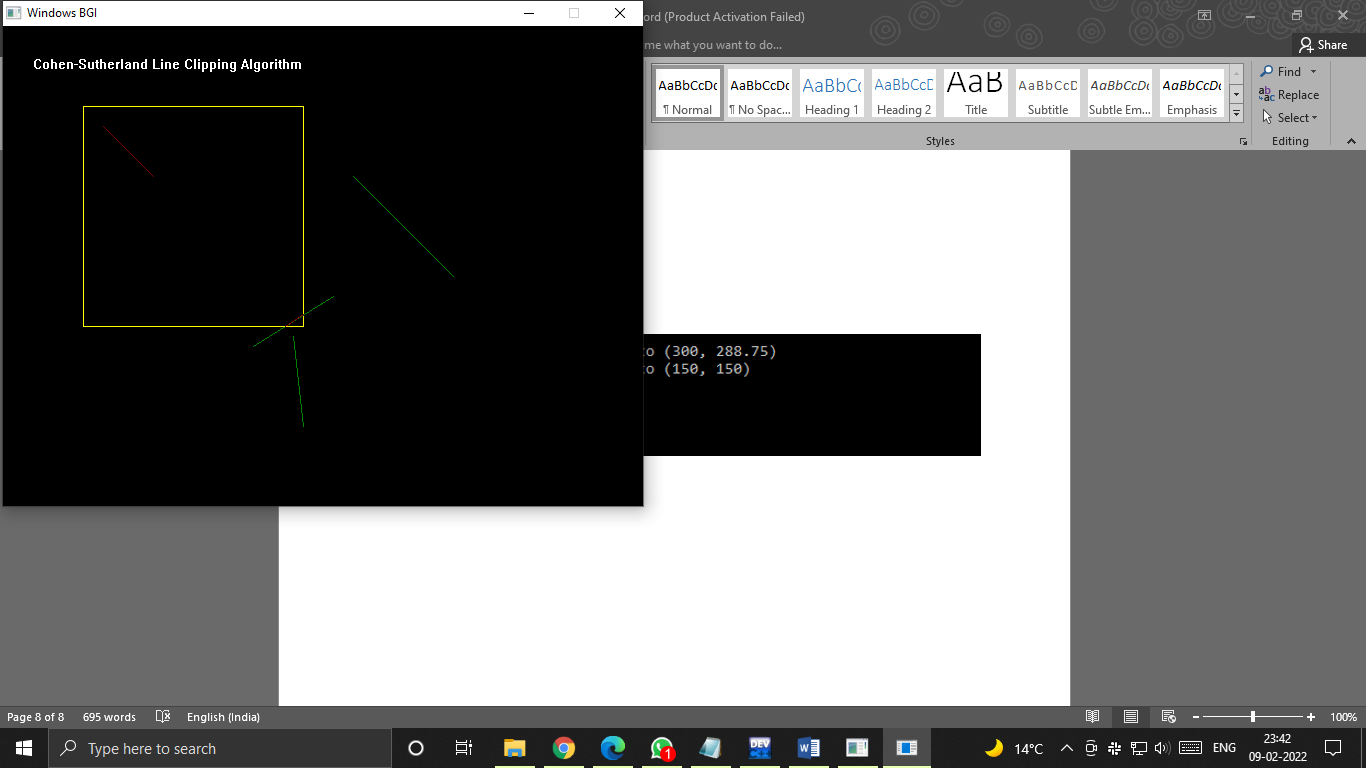
closegraph();

return 0;

}

**OUTPUT**





**Q6. Write a program to implement Sutherland Hodgeman Clipping program.**

#include<iostream>

#include<conio.h>

#include<graphics.h>

using namespace std;

#define round(a) ((int)(a+0.5))

int k;

float xmin,ymin,xmax,ymax,arr[20],m;

void clipl(float x1,float y1,float x2,float y2)

{

if(x2-x1)

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

else

m=100000;

if(x1 >= xmin && x2 >= xmin)

{

arr[k]=x2;

arr[k+1]=y2;

k+=2;

}

if(x1 < xmin && x2 >= xmin)

{

arr[k]=xmin;

arr[k+1]=y1+m\*(xmin-x1);

arr[k+2]=x2;

arr[k+3]=y2;

k+=4;

}

if(x1 >= xmin && x2 < xmin)

{

arr[k]=xmin;

arr[k+1]=y1+m\*(xmin-x1);

k+=2;

}

}

void clipt(float x1,float y1,float x2,float y2)

{

if(y2-y1)

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

else

m=100000;

if(y1 <= ymax && y2 <= ymax)

{

arr[k]=x2;

arr[k+1]=y2;

k+=2;

}

if(y1 > ymax && y2 <= ymax)

{

arr[k]=x1+m\*(ymax-y1);

arr[k+1]=ymax;

arr[k+2]=x2;

arr[k+3]=y2;

k+=4;

}

if(y1 <= ymax && y2 > ymax)

{

arr[k]=x1+m\*(ymax-y1);

arr[k+1]=ymax;

k+=2;

}

}

void clipr(float x1,float y1,float x2,float y2)

{

if(x2-x1)

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

else

m=100000;

if(x1 <= xmax && x2 <= xmax)

{

arr[k]=x2;

arr[k+1]=y2;

k+=2;

}

if(x1 > xmax && x2 <= xmax)

{

arr[k]=xmax;

arr[k+1]=y1+m\*(xmax-x1);

arr[k+2]=x2;

arr[k+3]=y2;

k+=4;

}

if(x1 <= xmax && x2 > xmax)

{

arr[k]=xmax;

arr[k+1]=y1+m\*(xmax-x1);

k+=2;

}

}

void clipb(float x1,float y1,float x2,float y2)

{

if(y2-y1)

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

else

m=100000;

if(y1 >= ymin && y2 >= ymin)

{

arr[k]=x2;

arr[k+1]=y2;

k+=2;

}

if(y1 < ymin && y2 >= ymin)

{

arr[k]=x1+m\*(ymin-y1);

arr[k+1]=ymin;

arr[k+2]=x2;

arr[k+3]=y2;

k+=4;

}

if(y1 >= ymin && y2 < ymin)

{

arr[k]=x1+m\*(ymin-y1);

arr[k+1]=ymin;

k+=2;

}

}

int main()

{

int gd=DETECT,gm,n,poly[20];

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

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

cout<<"Coordinates of rectangular clip window :\nxmin,ymin :";

cin>>xmin>>ymin;

cout<<"xmax,ymax :";

cin>>xmax>>ymax;

cout<<"\n\nPolygon to be clipped :\nNumber of sides :";

cin>>n;

cout<<"Enter the coordinates :";

int i;

for(i=0;i < 2\*n;i++)

cin>>polyy[i];

polyy[i]=polyy[0];

polyy[i+1]=polyy[1];

for(i=0;i < 2\*n+2;i++)

poly[i]=round(polyy[i]);

setcolor(RED);

rectangle(xmin,ymax,xmax,ymin);

cout<<"\t\tUNCLIPPED POLYGON";

setcolor(WHITE);

fillpoly(n,poly);

getch();

cleardevice();

k=0;

for(i=0;i < 2\*n;i+=2)

clipl(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);

n=k/2;

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

polyy[i]=arr[i];

polyy[i]=polyy[0];

polyy[i+1]=polyy[1];

k=0;

for(i=0;i < 2\*n;i+=2)

clipt(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);

n=k/2;

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

polyy[i]=arr[i];

polyy[i]=polyy[0];

polyy[i+1]=polyy[1];

k=0;

for(i=0;i < 2\*n;i+=2)

clipr(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);

n=k/2;

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

polyy[i]=arr[i];

polyy[i]=polyy[0];

polyy[i+1]=polyy[1];

k=0;

for(i=0;i < 2\*n;i+=2)

clipb(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);

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

poly[i]=round(arr[i]);

if(k)

fillpoly(k/2,poly);

setcolor(RED);

rectangle(xmin,ymax,xmax,ymin);

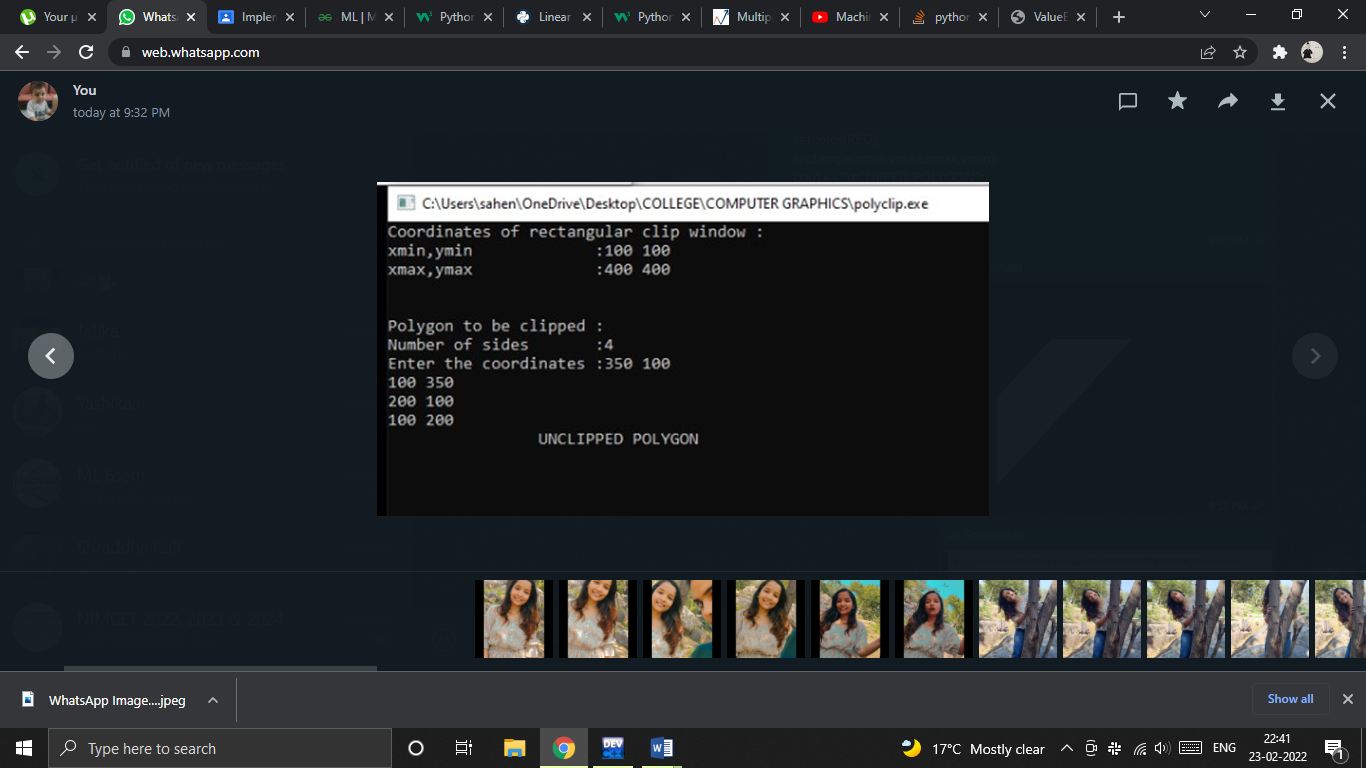
cout<<"\tCLIPPED POLYGON";

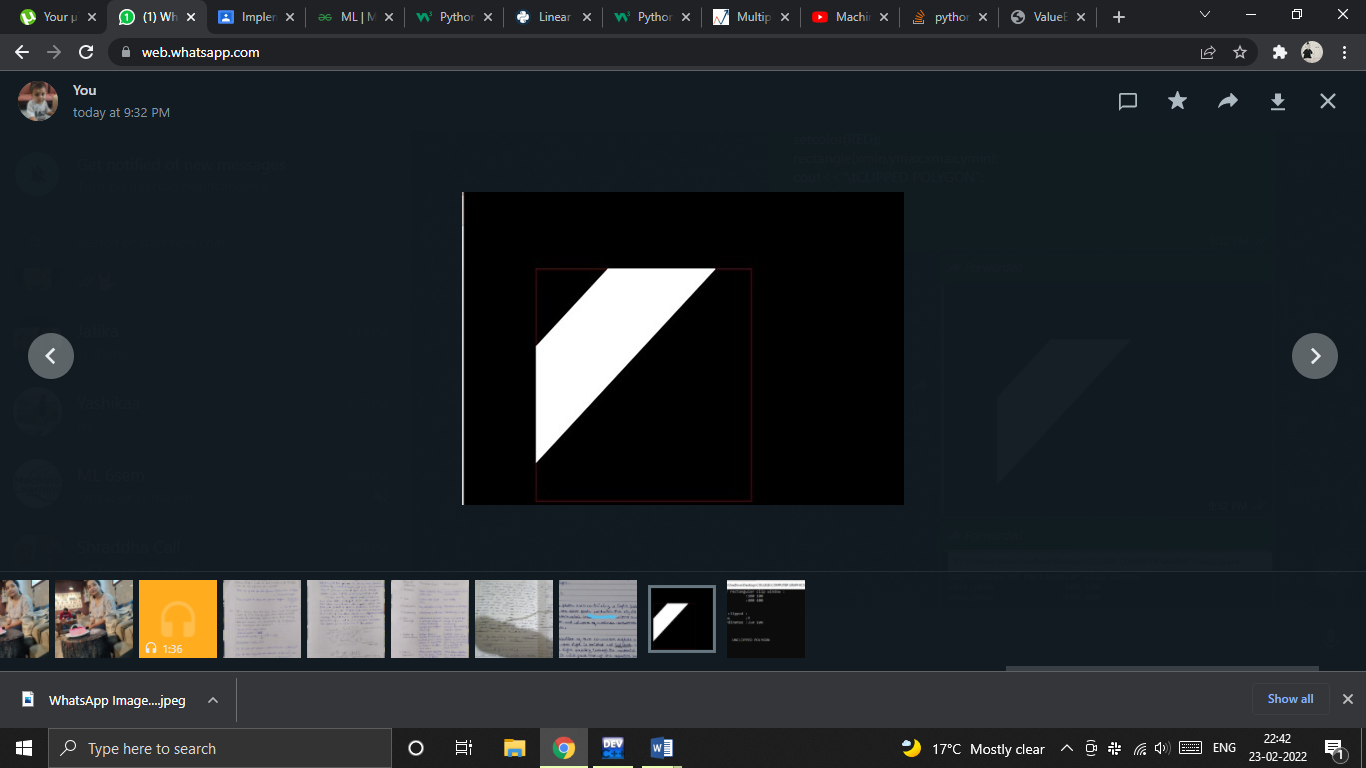
getch();

closegraph();

}

**OUTPUT**





**Q7. Write a program to implement Scan-Line Polygon fill algorithm.**

#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) // non-horizontal line

{

edge = (Edge \*) malloc (sizeof(Edge));

if (v1.y < v2.y) // up-going 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);

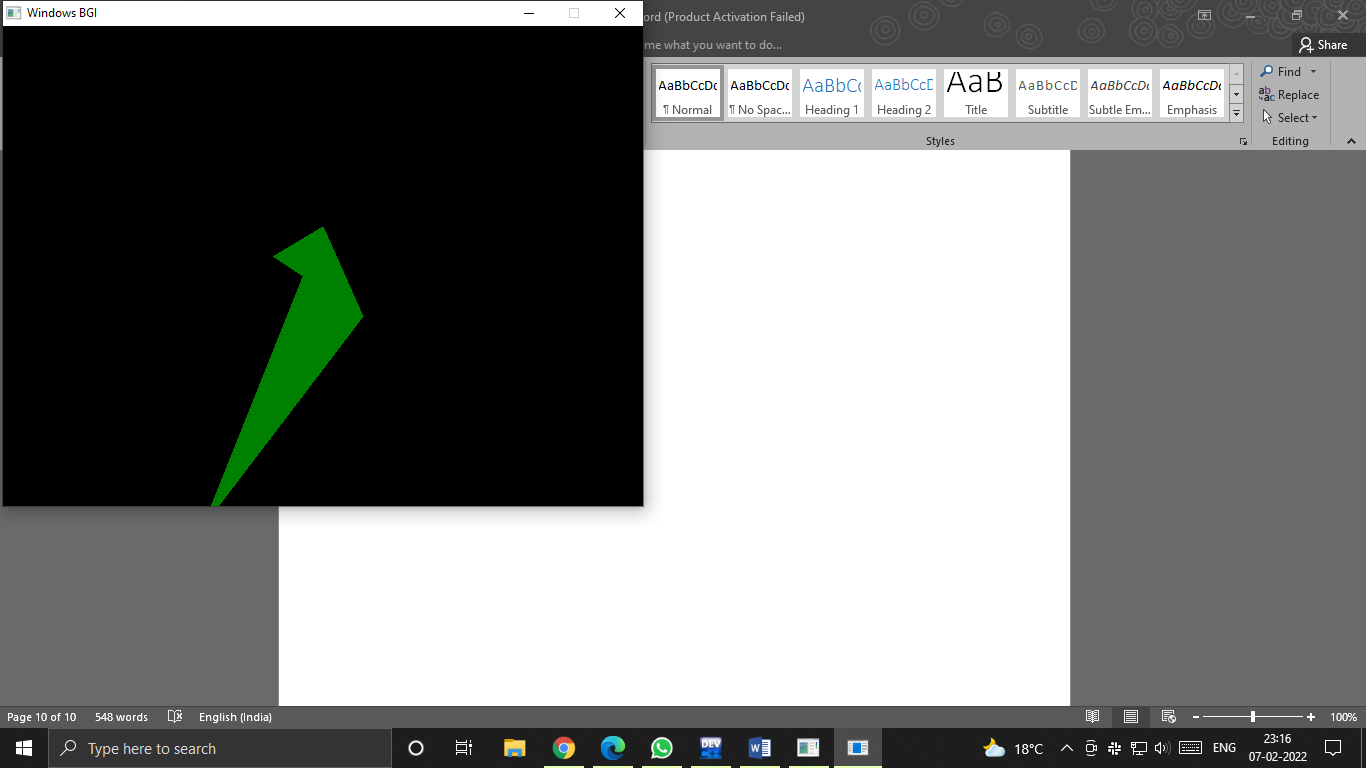
getch();

closegraph();

return 0;

}

**OUTPUT**



**Q8. Write a program to apply various 2D transformations on 2D object (use homogeneous objects).**

#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;

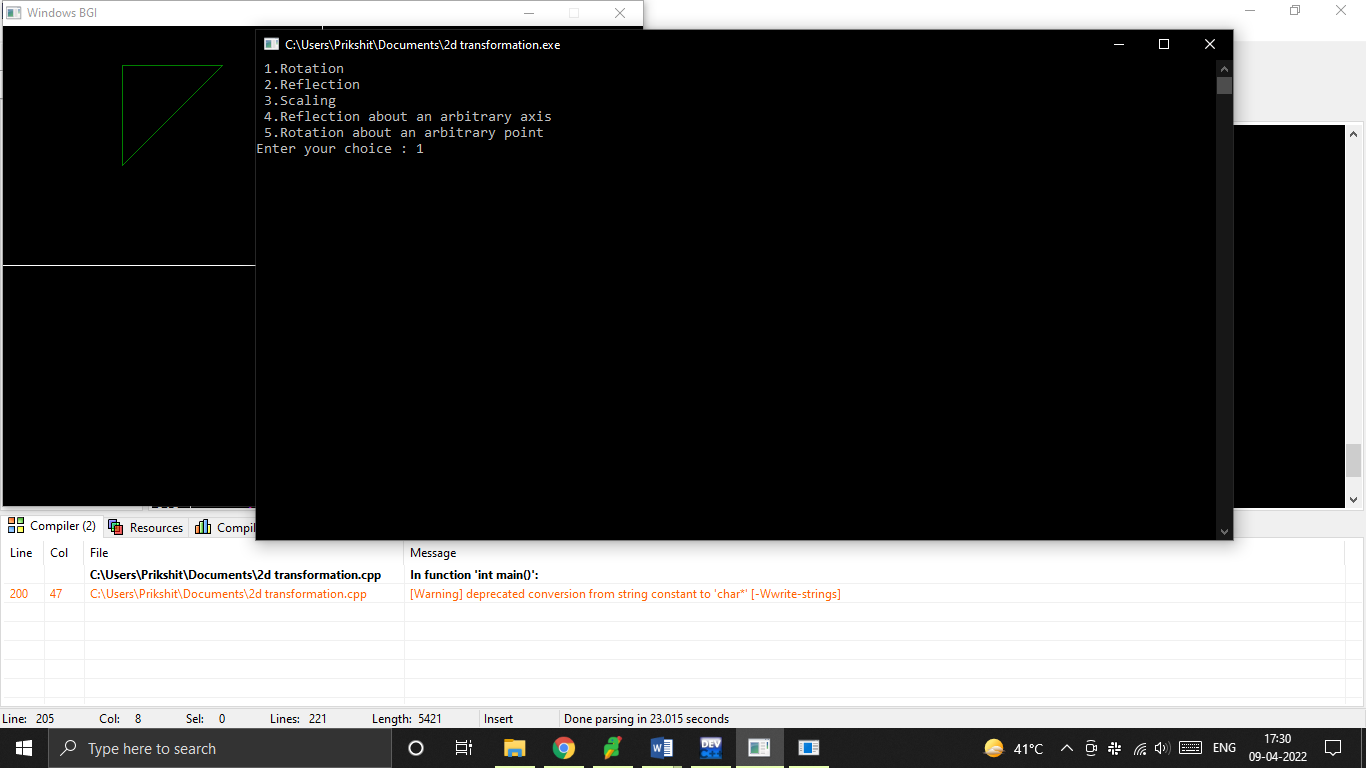
default : cout<<"Invalid Choice\n";

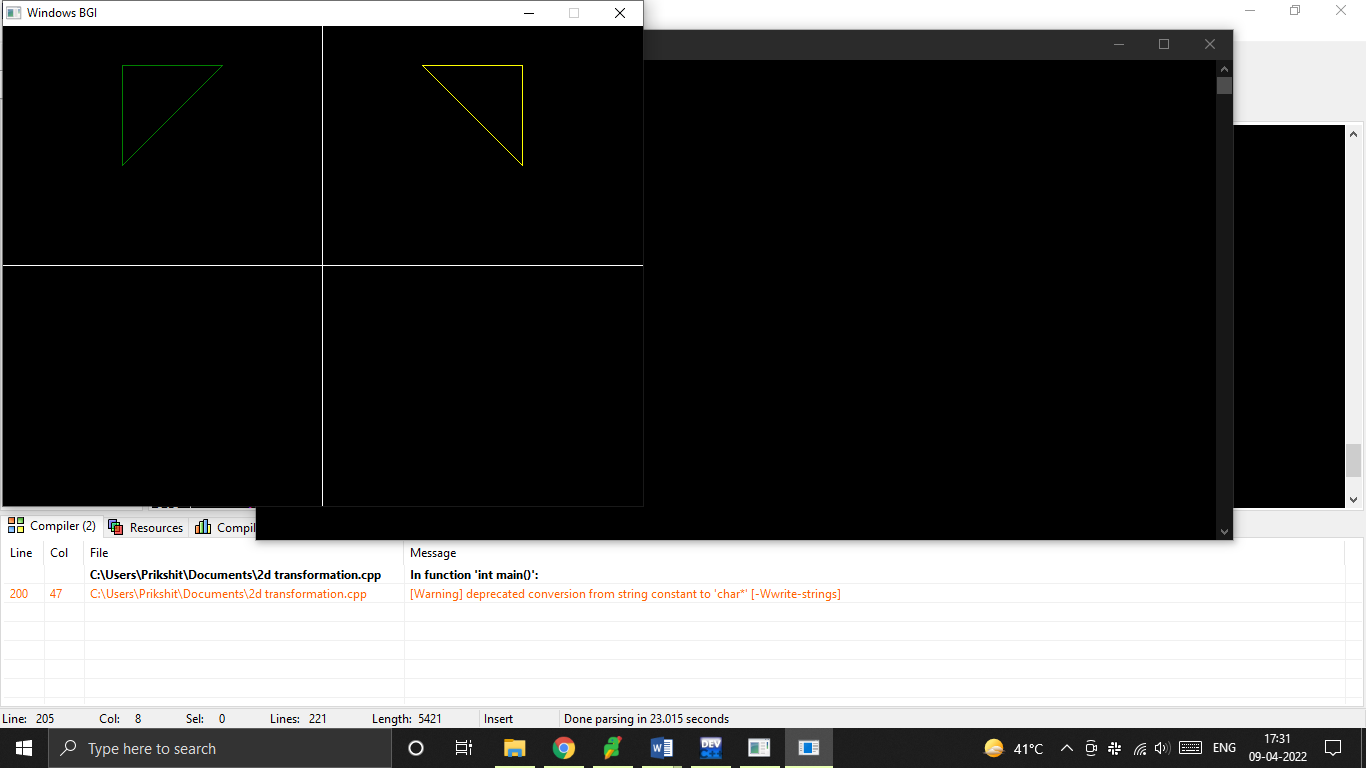
}

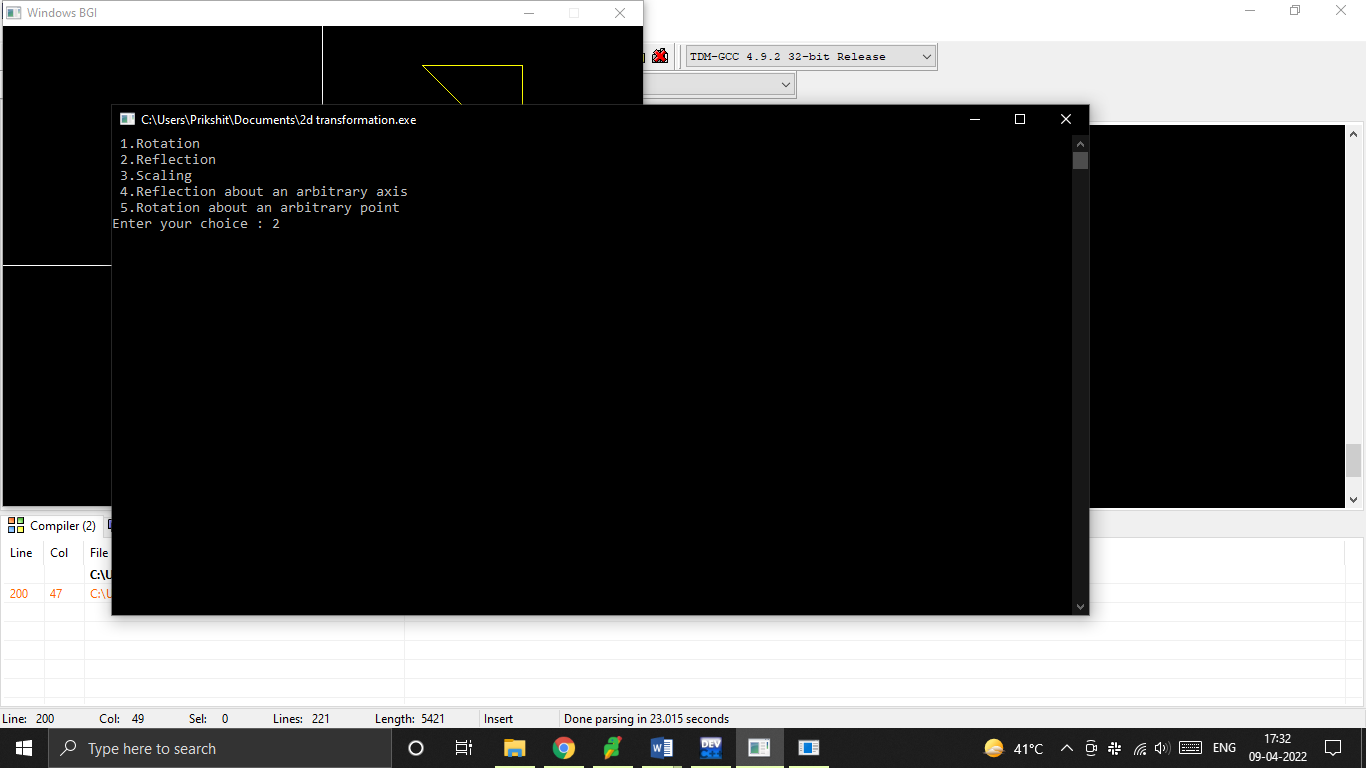
getch();

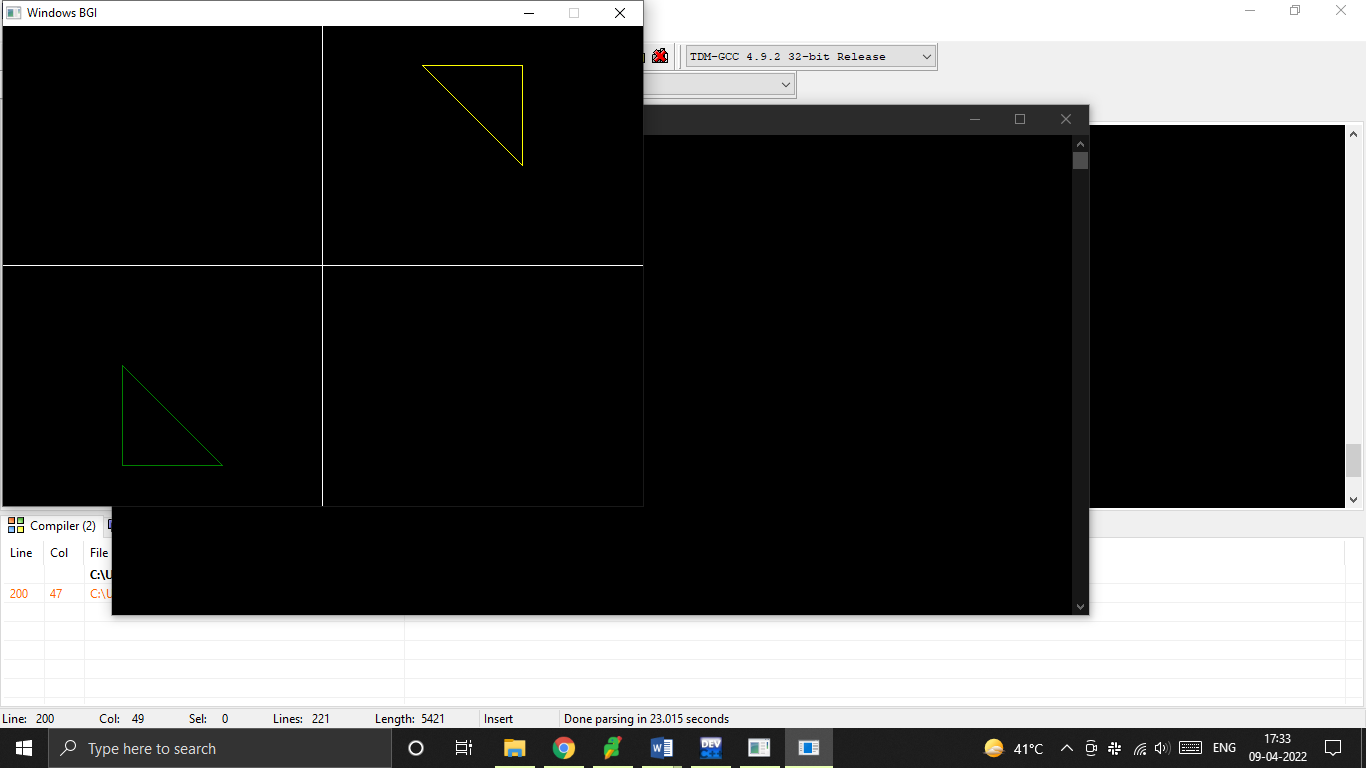
}

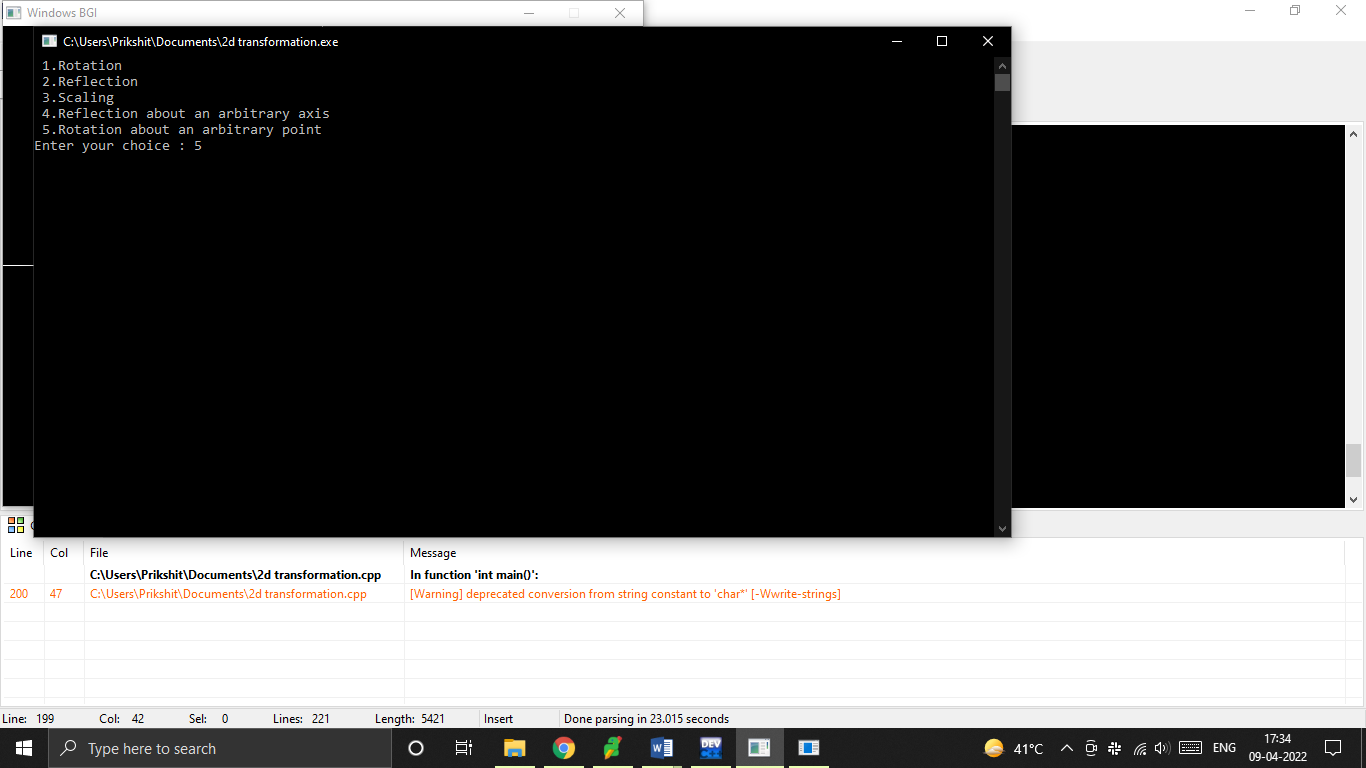
**OUTPUT**

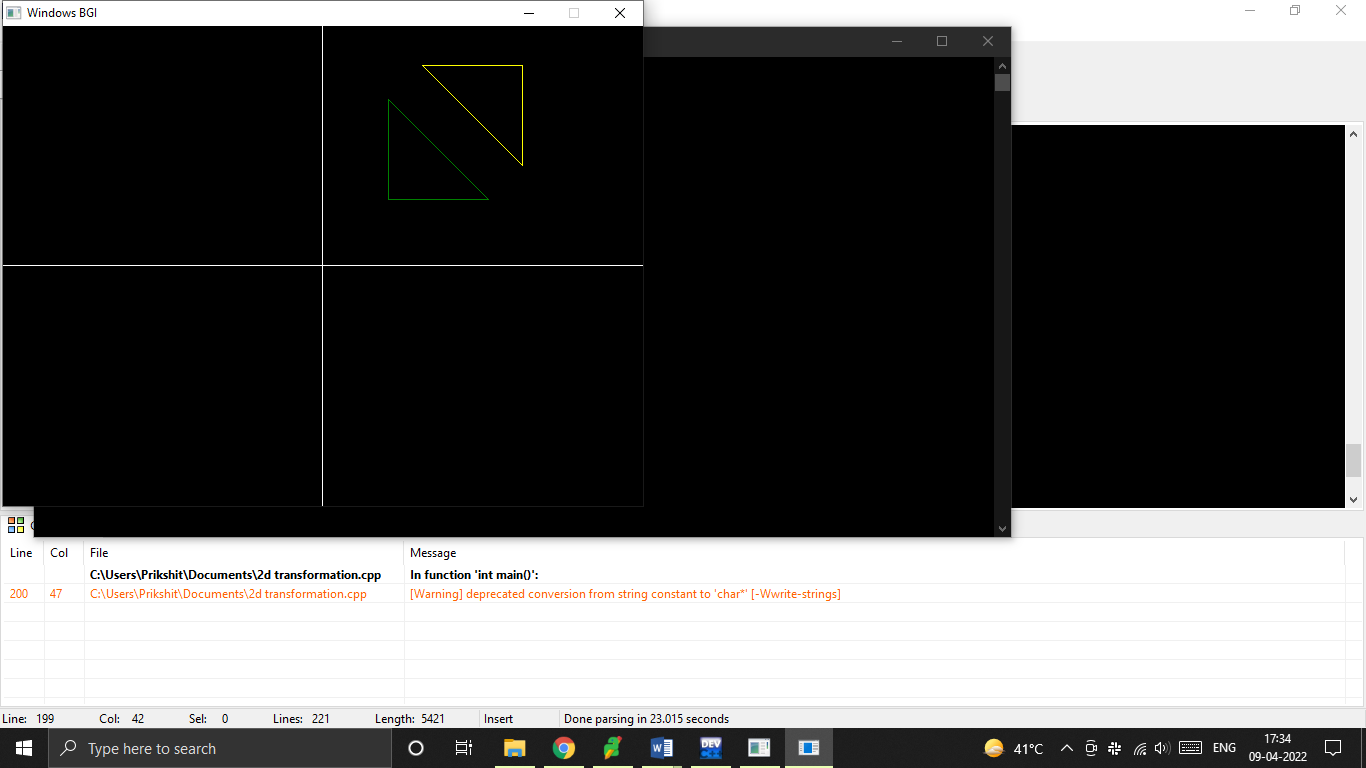












**Q9. Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it.**

#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 ";

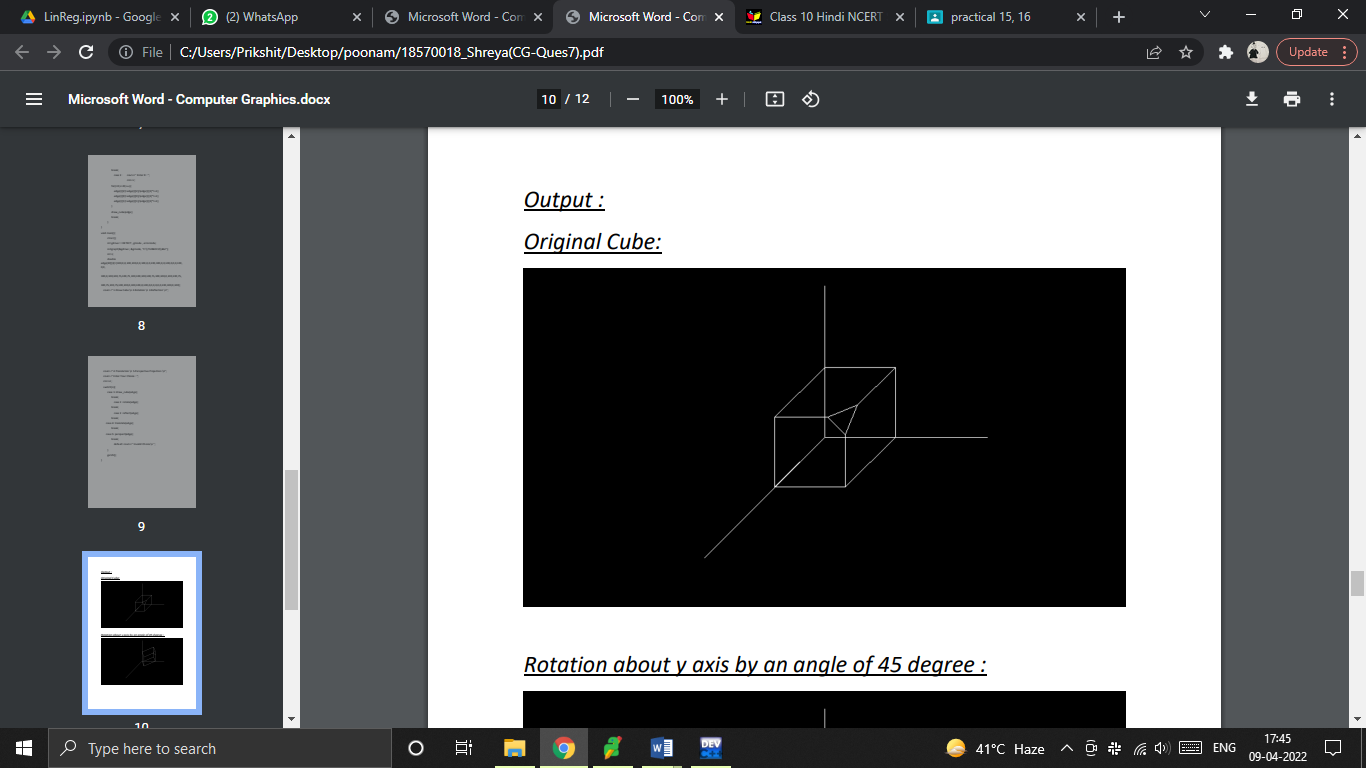
}

getch();

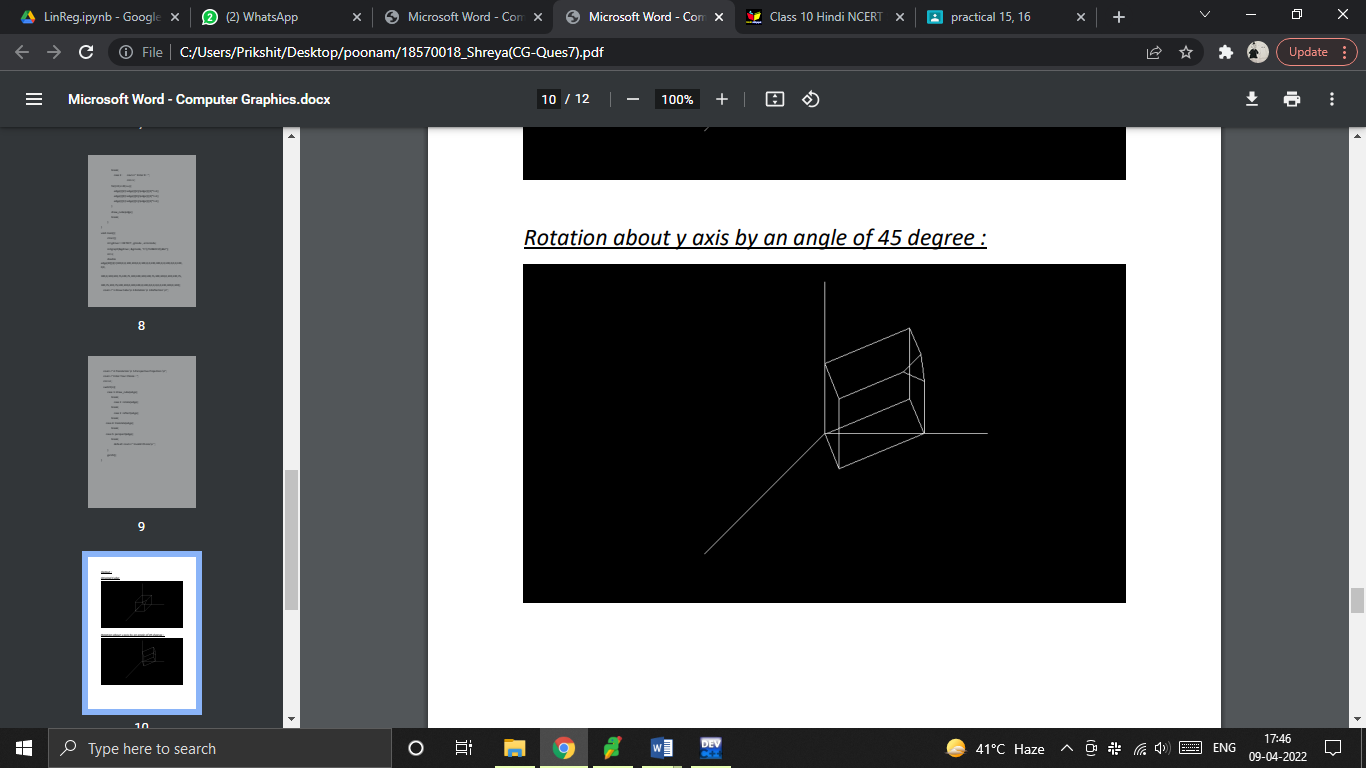
}

**OUTPUT**

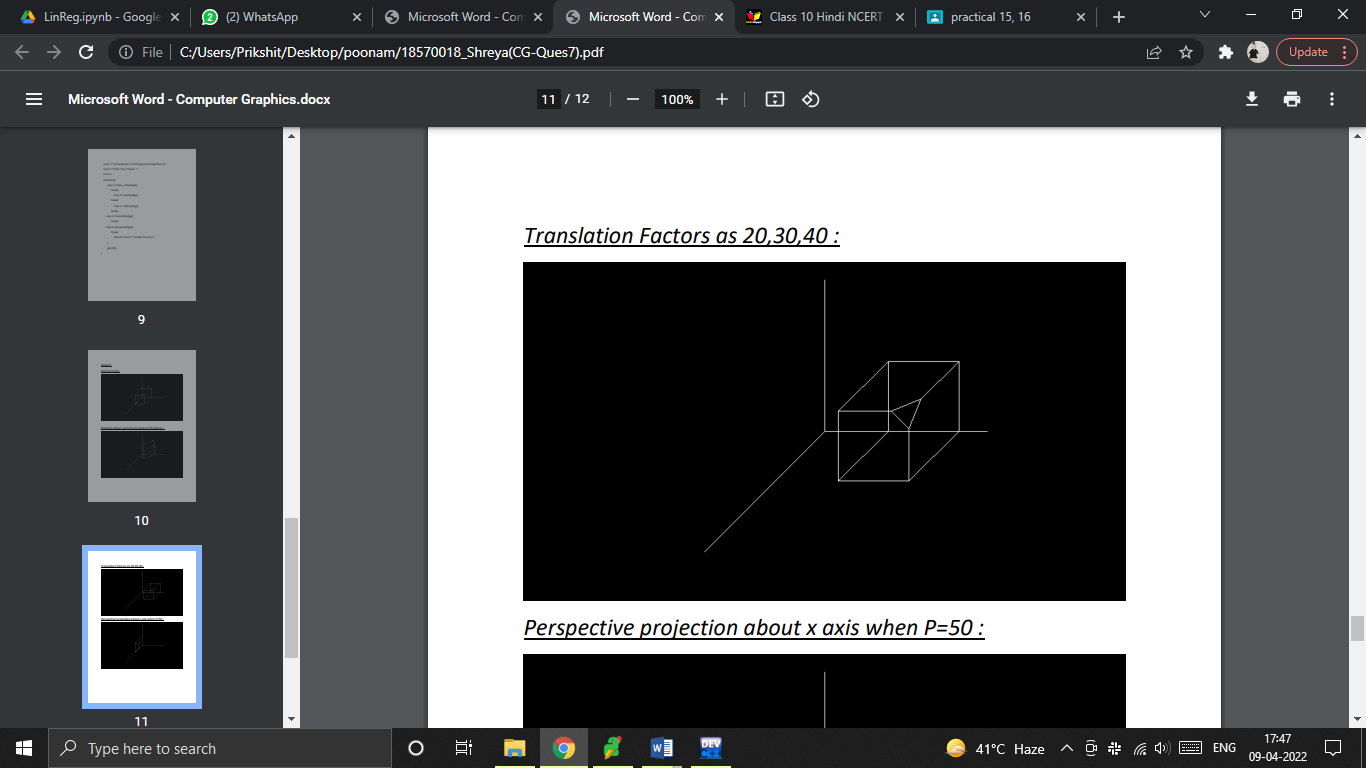
**ORIGINAL CUBE:**



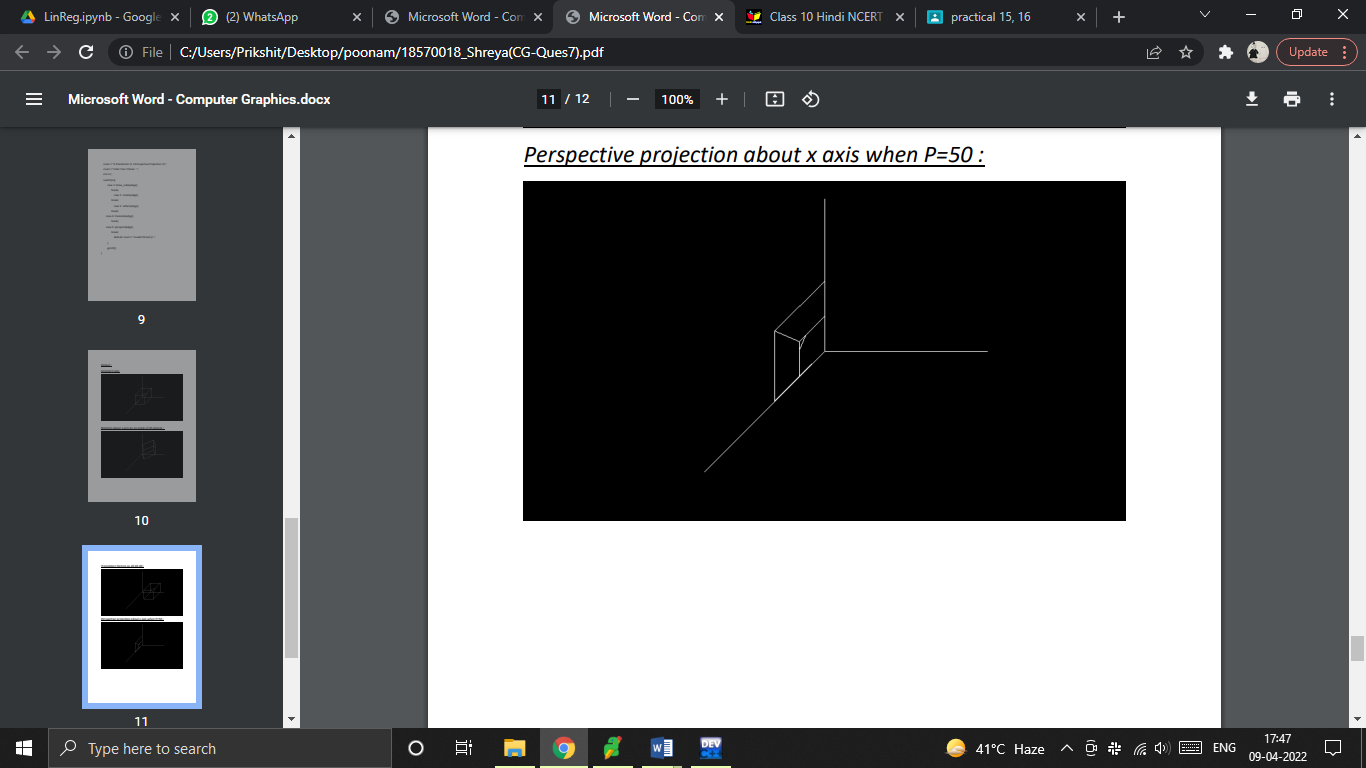
**ROTATION ABOUT Y-AXIS BY AN ANGLE OF 45 DEGREE:**



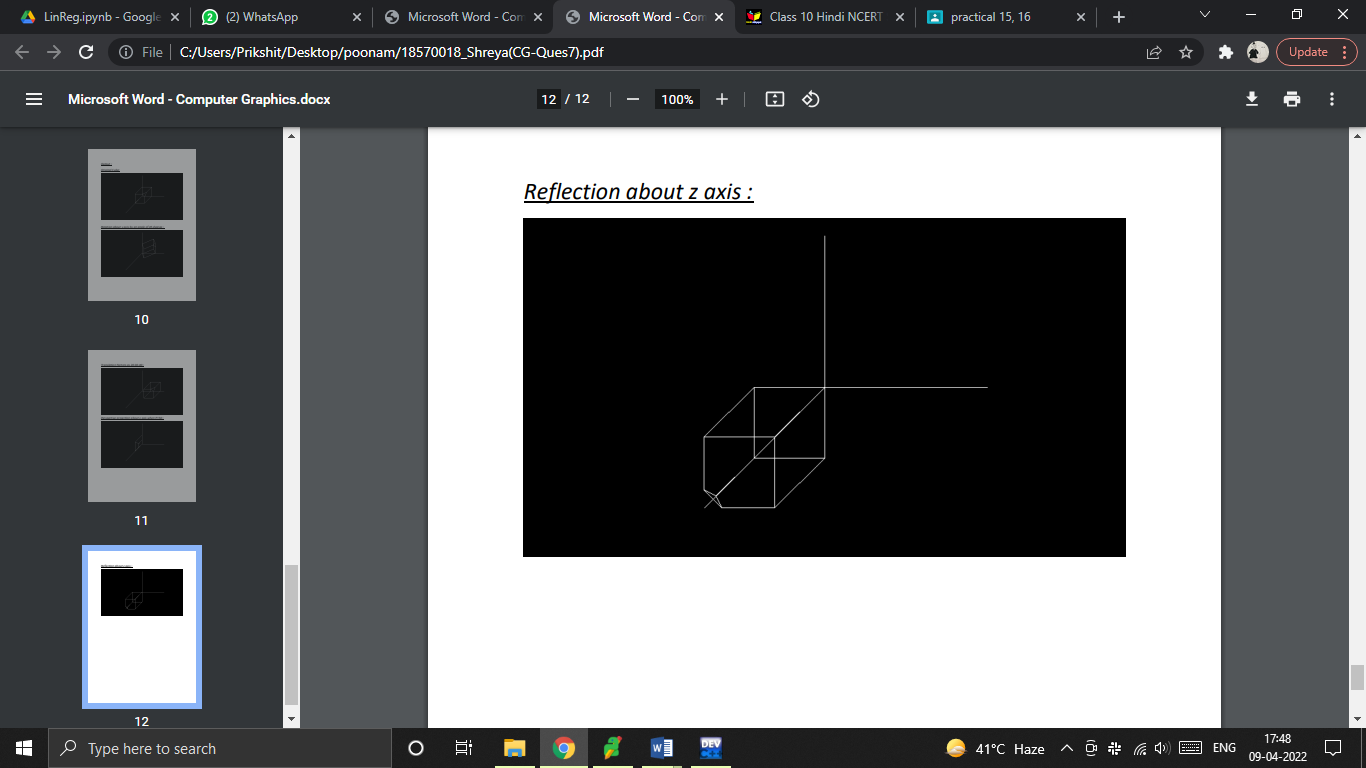
**TRANSLATION FACTORS AS 20, 30, 40:**



**PERSPECTIVE PROJECTION ABOUT X-AXIS WHEN P=50:**



**REFLECTION ABOUT Z-AXIS:**



**Q10. Write a program to draw Hermite/Bezier curve.**

#include<iostream>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void bezier\_curve(int x[4],int y[4]){

double t;

for(t=0.0;t<1.0;t=t+0.0005){

double xt=pow(1-t,3)\*x[0]+3\*t\*pow(1-

t,2)\*x[1]+3\*pow(t,2)\*(1-t)\*x[2]+pow(t,3)\*x[3];

double yt=pow(1-t,3)\*y[0]+3\*t\*pow(1-

t,2)\*y[1]+3\*pow(t,2)\*(1-t)\*y[2]+pow(t,3)\*y[3];

putpixel(xt,yt,YELLOW);

}

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

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

}

}

void hermite\_curve(int x1,int y1,int x2,int y2,double t1,double

t4){

float x,y,t;

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

x=(2\*t\*t\*t-3\*t\*t+1)\*x1+(-2\*t\*t\*t+3\*t\*t)\*x2+(t\*t\*t-

2\*t\*t+t)\*t1+(t\*t\*t-t\*t)\*t4;

y=(2\*t\*t\*t-3\*t\*t+1)\*y1+(-2\*t\*t\*t+3\*t\*t)\*y2+(t\*t\*t-

2\*t\*t+1)\*t1+(t\*t\*t-t\*t)\*t4;

putpixel(x,y,YELLOW);

}

putpixel(x1,y1,GREEN);

putpixel(x2,y2,GREEN);

line(x1,y1,x2,y2);

}

voidmain()

{

clrscr();

int gdriver=DETECT,gmode,errorcode;

int x1,y1,x2,y2,n;

double t1,t4;

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

int x[4],y[4];

int i;

cout<<"1.BezierCurve\n2.HermiteCurve\n";

cout<<"Enteryourchoice:";

cin>>n;

if(n==1){

cout<<"Enterxandycoordinates\n";

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

cout<<"x"<<i+1<<":";

cin>>x[i];

cout<<"y"<<i+1<<":";

cin>>y[i];

cout<<endl;

}

bezier\_curve(x,y);

}

elseif(n==2){

cout<<"Enterthexcoordinateof1sthermitepoint:";

cin>>x1;

cout<<"Entertheycoordinateof1sthermitepoint:";

cin>>y1;

cout<<"Enterthexcoordinateof4thhermitepoint:";

cin>>x2;

cout<<"Entertheycoordinateof4thhermitepoint:";

cin>>y2;

cout<<"Entertangentatp1:";

cin>>t1;

cout<<"Entertangentatp4:";

cin>>t4;

hermite\_curve(x1,y1,x2,y2,t1,t4);

}

else{

cout<<"\n InvalidChoice";

}

getch();

}

**OUPUT**

