**Question 1**

#include <iostream>

#include <graphics.h>

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

int dx = abs(x2 - x1);

int dy = abs(y2 - y1);

int sx, sy;

if (x1 < x2) {

sx = 1;

} else {

sx = -1;

}

if (y1 < y2) {

sy = 1;

} else {

sy = -1;

}

int err = dx - dy;

int x = x1;

int y = y1;

while (x != x2 || y != y2) {

putpixel(x, y, WHITE);

int e2 = 2 \* err;

if (e2 > -dy) {

err = err - dy;

x = x + sx;

}

if (e2 < dx) {

err = err + dx;

y = y + sy;

}

}

}

int main() {

int gd = DETECT, gm;

initgraph(&gd, &gm, "");

int x1 = 100, y1 = 100, x2 = 300, y2 = 300;

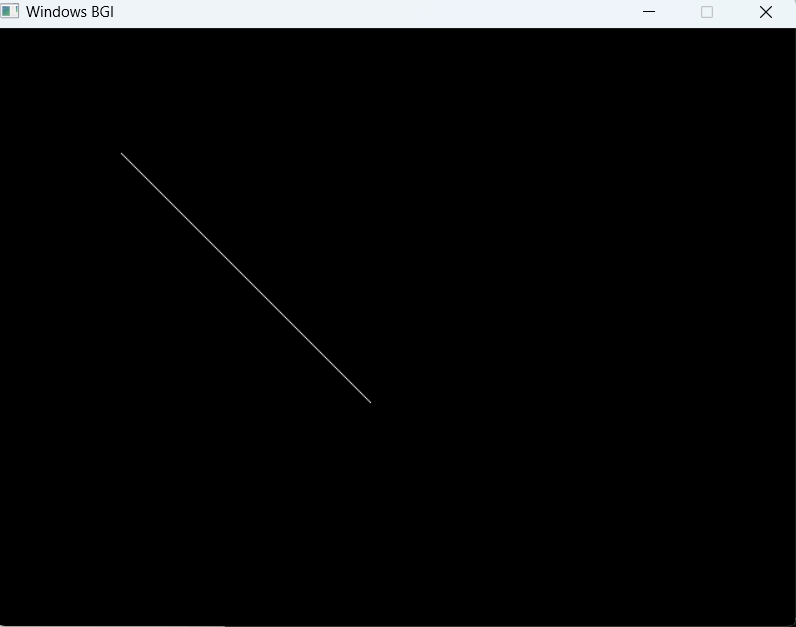
drawLine(x1, y1, x2, y2);

getch();

closegraph();

return 0;

}

****

**Question 2**

#include <iostream>

#include <graphics.h>

void drawCircle(int x0, int y0, int radius) {

int x = radius;

int y = 0;

int err = 0;

while (x >= y) {

putpixel(x0 + x, y0 + y, WHITE);

putpixel(x0 + y, y0 + x, WHITE);

putpixel(x0 - y, y0 + x, WHITE);

putpixel(x0 - x, y0 + y, WHITE);

putpixel(x0 - x, y0 - y, WHITE);

putpixel(x0 - y, y0 - x, WHITE);

putpixel(x0 + y, y0 - x, WHITE);

putpixel(x0 + x, y0 - y, WHITE);

if (err <= 0) {

y += 1;

err += 2 \* y + 1;

}

if (err > 0) {

x -= 1;

err -= 2 \* x + 1;

}

}

}

int main() {

int gd = DETECT, gm;

initgraph(&gd, &gm, "");

int x0 = 200, y0 = 200, radius = 100;

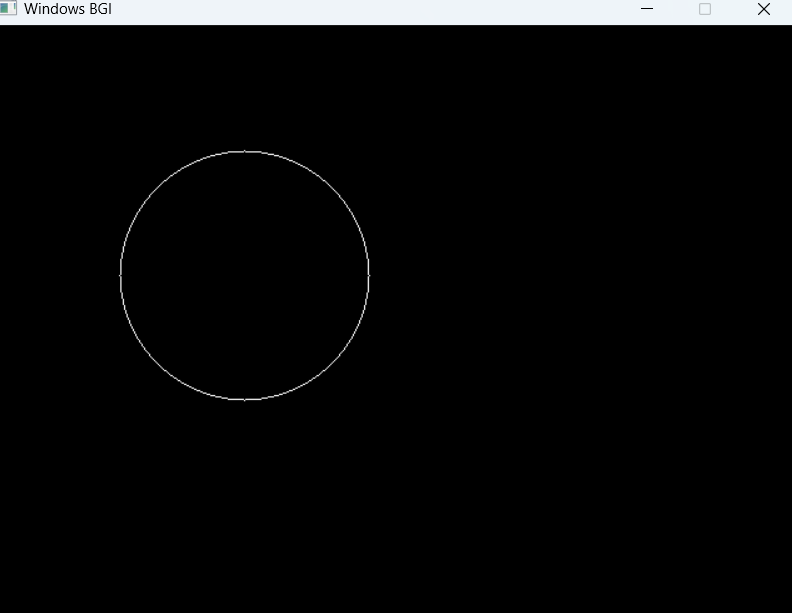
drawCircle(x0, y0, radius);

getch();

closegraph();

return 0;

}



**Question3**

#include<graphics.h>

#include <iostream>

using namespace std;

int main()

{

int rcode\_begin[4]={0,0,0,0},rcode\_end[4]={0,0,0,0},region\_code[4];

int W\_xmax,W\_ymax,W\_xmin,W\_ymin,flag=0;

float slope;

int x,y,x1,y1,i, xc,yc;

int gr=DETECT,gm;

initgraph(&gr,&gm," ");

cout<<"\n\*\*\*\*\*\* Cohen Sutherlsnd Line Clipping algorithm \*\*\*\*\*\*\*\*\*\*\*";

cout<<"\n Now, enter XMin, YMin =";

cin>>W\_xmin>>W\_ymin;

cout<<"\n First enter XMax, YMax =";

cin>>W\_xmax>>W\_ymax;

cout<<"\n Please enter intial point x and y= ";

cin>>x>>y;

cout<<"\n Now, enter final point x1 and y1= ";

cin>>x1>>y1;

cleardevice();

rectangle(W\_xmin,W\_ymin,W\_xmax,W\_ymax);

line(x,y,x1,y1);

line(0,0,600,0);

line(0,0,0,600);

if(y>W\_ymax) {

rcode\_begin[0]=1; // Top

flag=1 ;

}

if(y<W\_ymin) {

rcode\_begin[1]=1; // Bottom

flag=1;

}

if(x>W\_xmax) {

rcode\_begin[2]=1; // Right

flag=1;

}

if(x<W\_xmin) {

rcode\_begin[3]=1; //Left

flag=1;

}

//end point of Line

if(y1>W\_ymax){

rcode\_end[0]=1; // Top

flag=1;

}

if(y1<W\_ymin) {

rcode\_end[1]=1; // Bottom

flag=1;

}

if(x1>W\_xmax){

rcode\_end[2]=1; // Right

flag=1;

}

if(x1<W\_xmin){

rcode\_end[3]=1; //Left

flag=1;

}

if(flag==0)

{

cout<<"No need of clipping as it is already in window";

}

flag=1;

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

region\_code[i]= rcode\_begin[i] && rcode\_end[i] ;

if(region\_code[i]==1)

flag=0;

}

if(flag==0)

{

cout<<"\n Line is completely outside the window";

}

else{

slope=(float)(y1-y)/(x1-x);

if(rcode\_begin[2]==0 && rcode\_begin[3]==1) //left

{

y=y+(float) (W\_xmin-x)\*slope ;

x=W\_xmin;

}

if(rcode\_begin[2]==1 && rcode\_begin[3]==0) // right

{

y=y+(float) (W\_xmax-x)\*slope ;

x=W\_xmax;

}

if(rcode\_begin[0]==1 && rcode\_begin[1]==0) // top

{

x=x+(float) (W\_ymax-y)/slope ;

y=W\_ymax;

}

if(rcode\_begin[0]==0 && rcode\_begin[1]==1) // bottom

{

x=x+(float) (W\_ymin-y)/slope ;

y=W\_ymin;

}

// end points

if(rcode\_end[2]==0 && rcode\_end[3]==1) //left

{

y1=y1+(float) (W\_xmin-x1)\*slope ;

x1=W\_xmin;

}

if(rcode\_end[2]==1 && rcode\_end[3]==0) // right

{

y1=y1+(float) (W\_xmax-x1)\*slope ;

x1=W\_xmax;

}

if(rcode\_end[0]==1 && rcode\_end[1]==0) // top

{

x1=x1+(float) (W\_ymax-y1)/slope ;

y1=W\_ymax;

}

if(rcode\_end[0]==0 && rcode\_end[1]==1) // bottom

{

x1=x1+(float) (W\_ymin-y1)/slope ;

y1=W\_ymin;

}

}

delay(1000);

clearviewport();

rectangle(W\_xmin,W\_ymin,W\_xmax,W\_ymax);

line(0,0,600,0);

line(0,0,0,600);

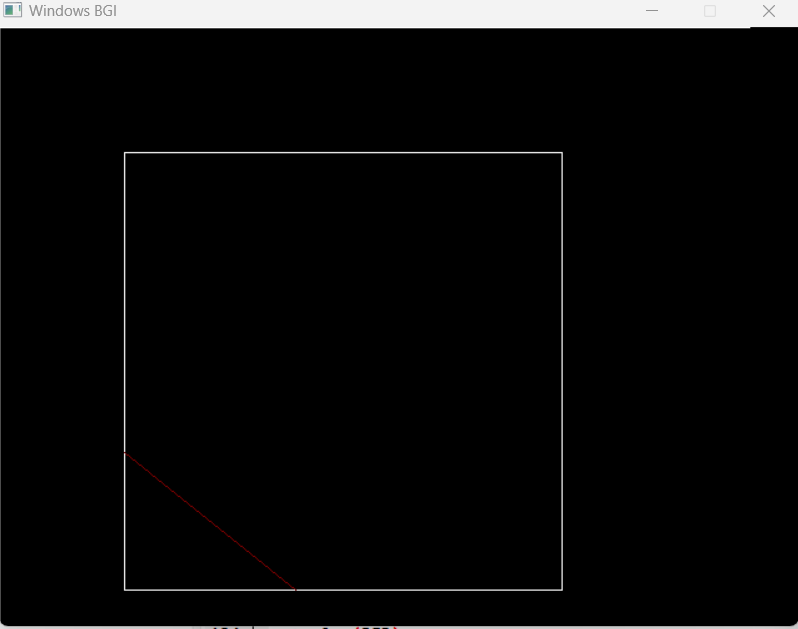
setcolor(RED);

line(x,y,x1,y1);

getch();

closegraph();

}



Question 4

#include<graphics.h>

#include<conio.h>

#include<iostream>

using namespace std;

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

int k;

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

void clip\_left(float x1, float y1, float x2, float y2)

{

if(x2-x1)

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

else

m=100000;

if(x1 >= xmin && x2 >=xmin) //lies inside the window

{

arr[k]=x2;

arr[k+1]=y2;

k+=2;

}

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

{ //partially inside

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 clip\_top(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 clip\_rt(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 clip\_btm(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;

int n,poly[20],i,k;

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

//clrscr();

cout<<"\n\t Enter the coordinates of the rectangular clip window ::-";

cout<<"\n\t xmin , ymin :: ";

cin>>xmin>>ymin;

cout<<"\n\t xmax, ymax :: ";

cin>>xmax>>ymax;

cout<<"\n\t Polygon to be clippped ::-";

cout<<"\n\t Number of Sides :: ";

cin>>n;

cout<<"\n\t Enter the coordinates of the ploygon :: \t";

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]);

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

setcolor(RED);

rectangle(xmin,ymax,xmax,ymin);

cout<<"\n\t Unclipped Polygon :: ";

setcolor(YELLOW);

fillpoly(n,poly);

getch();

cleardevice();

k=0;

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

clip\_left(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)

clip\_top(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];

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

k=0;

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

clip\_rt(polyy[i],polyy[i+1],polyy[i+2],poly[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)

clip\_btm(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(BLUE);

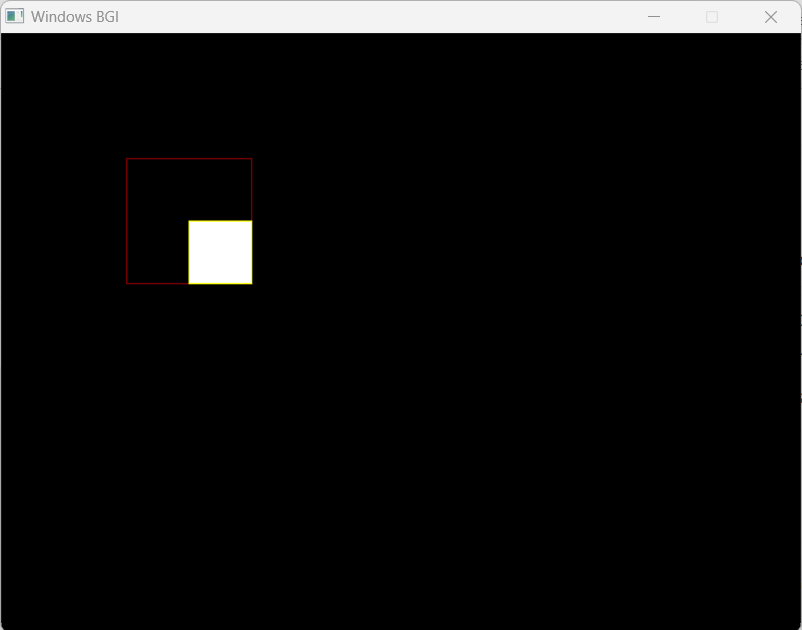
rectangle(xmin,ymax,xmax,ymin);

cout<<"\n\t Clipped Polygon :: ";

getch();

closegraph();

}

****

**Question 5**

#include <conio.h>

#include <iostream>

#include <graphics.h>

#include <stdlib.h>

using namespace std;

//Declaration of class point

class point

{

public:

int x,y;

};

class poly

{

private:

point p[20];

int inter[20],x,y;

int v,xmin,ymin,xmax,ymax;

public:

int c;

void read();

void calcs();

void display();

void ints(float);

void sort(int);

};

void poly::read()

{

int i;

cout<<"\n\t SCAN\_FILL ALGORITHM";

cout<<"\n Enter the no of vertices of polygon:";

cin>>v;

if(v>2)

{

for(i=0;i<v; i++) //ACCEPT THE VERTICES

{

cout<<"\nEnter the co-ordinate no.- "<<i+1<<" : ";

cout<<"\n\tx"<<(i+1)<<"=";

cin>>p[i].x;

cout<<"\n\ty"<<(i+1)<<"=";

cin>>p[i].y;

}

p[i].x=p[0].x;

p[i].y=p[0].y;

xmin=xmax=p[0].x;

ymin=ymax=p[0].y;

}

else

cout<<"\n Enter valid no. of vertices.";

}

//FUNCTION FOR FINDING

void poly::calcs()

{ //MAX,MIN

for(int i=0;i<v;i++)

{

if(xmin>p[i].x)

xmin=p[i].x;

if(xmax<p[i].x)

xmax=p[i].x;

if(ymin>p[i].y)

ymin=p[i].y;

if(ymax<p[i].y)

ymax=p[i].y;

}

}

//DISPLAY FUNCTION

void poly::display()

{

int ch1;

char ch='y';

float s,s2;

do

{

cout<<"\n\nMENU:";

cout<<"\n\n\t1 . Scan line Fill ";

cout<<"\n\n\t2 . Exit ";

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

cin>>ch1;

switch(ch1)

{

case 1:

s=ymin+0.01;

delay(100);

cleardevice();

while(s<=ymax)

{

ints(s);

sort(s);

s++;

}

break;

case 2:

exit(0);

}

cout<<"Do you want to continue?: ";

cin>>ch;

}while(ch=='y' || ch=='Y');

}

void poly::ints(float z) //DEFINE FUNCTION INTS

{

int x1,x2,y1,y2,temp;

c=0;

for(int i=0;i<v;i++)

{

x1=p[i].x;

y1=p[i].y;

x2=p[i+1].x;

y2=p[i+1].y;

if(y2<y1)

{

temp=x1;

x1=x2;

x2=temp;

temp=y1;

y1=y2;

y2=temp;

}

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

{

if((y1-y2)==0)

x=x1;

else // used to make changes in x. so that we can fill our polygon after cerain distance

{

x=((x2-x1)\*(z-y1))/(y2-y1);

x=x+x1;

}

if(x<=xmax && x>=xmin)

inter[c++]=x;

}

}

}

void poly::sort(int z) //SORT FUNCTION

{

int temp,j,i;

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

{

line(p[i].x,p[i].y,p[i+1].x,p[i+1].y); // used to make hollow outlines of a polygon

}

delay(100);

for(i=0; i<c;i+=2)

{

delay(100);

line(inter[i],z,inter[i+1],z); // Used to fill the polygon ....

}

}

int main() //START OF MAIN

{

int cl;

initwindow(500,600);

cleardevice();

poly x;

x.read();

x.calcs();

cleardevice();

cout<<"\n\tEnter the colour u want:(0-15)->"; //Selecting colour

cin>>cl;

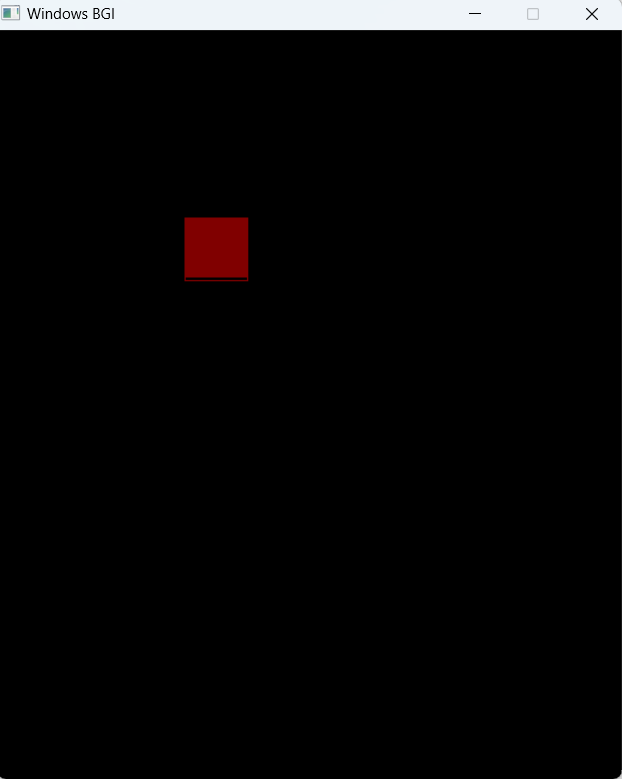
setcolor(cl);

x.display();

closegraph(); //CLOSE OF GRAPH

getch();

}



**Question6**

#include<iostream>

#include<graphics.h>

#include<math.h>

using namespace std;

int main()

{

int gd=DETECT,gm,s;

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

cout<<"1.Translation\n2.Rotation\n3.Scaling\n "<<endl;

cout<<"Selection:";

cin>>s;

switch(s)

{

case 1:

{

int x1,y1,x2,y2;

int x,y;

cout<<"Enter coordinates of polygon : \n";

cout<<"x1 : ";

cin>>x1;

cout<<"y1 : ";

cin>>y1;

cout<<"x2 : ";

cin>>x2;

cout<<"y2 : ";

cin>>y2;

cout<<"Enter coordinates of polygon for translation : \n";

cout<<"x : ";

cin>>x;

cout<<"y : ";

cin>>y;

cout<<"Rectangle before translation"<<endl;

setcolor(3);

rectangle(x1,y1,x2,y2);

setcolor(4);

cout<<"Rectangle after translation"<<endl;

rectangle(x1+x,y1+y,x2+x,y2+y);

getch();

break;

}

case 2:

{

long x1,y1,x2,y2;

int x,y;

cout<<"Enter coordinates of polygon : \n";

cout<<"x1 : ";

cin>>x1;

cout<<"y1 : ";

cin>>y1;

cout<<"x2 : ";

cin>>x2;

cout<<"y2 : ";

cin>>y2;

double a;

cout<<"Rectangle with rotation"<<endl;

setcolor(3);

rectangle(x1,y1,x2,y2);

cout<<"Angle of rotation:";

cin>>a;

a=(a\*3.14)/180;

long xr=x1+((x2-x1)\*cos(a)-(y2-y1)\*sin(a));

long yr=y1+((x2-x1)\*sin(a)+(y2-y1)\*cos(a));

setcolor(2);

rectangle(x1,y1,xr,yr);

getch();

break;

}

case 3:

{

int x1,y1,x2,y2;

int x,y;

cout<<"Enter coordinates of polygon : \n";

cout<<"x1 : ";

cin>>x1;

cout<<"y1 : ";

cin>>y1;

cout<<"x2 : ";

cin>>x2;

cout<<"y2 : ";

cin>>y2;

cout<<"Enter coordinates of polygon for scaling : \n";

cout<<"x : ";

cin>>x;

cout<<"y : ";

cin>>y;

cout<<"Before scaling"<<endl;

setcolor(3);

rectangle(x1,y1,x2,y2);

cout<<"After scaling"<<endl;

setcolor(10);

rectangle(x1\*x,y1\*y,x2\*x,y2\*y);

getch();

break;

}

default:

{

cout<<"Invalid Selection"<<endl;

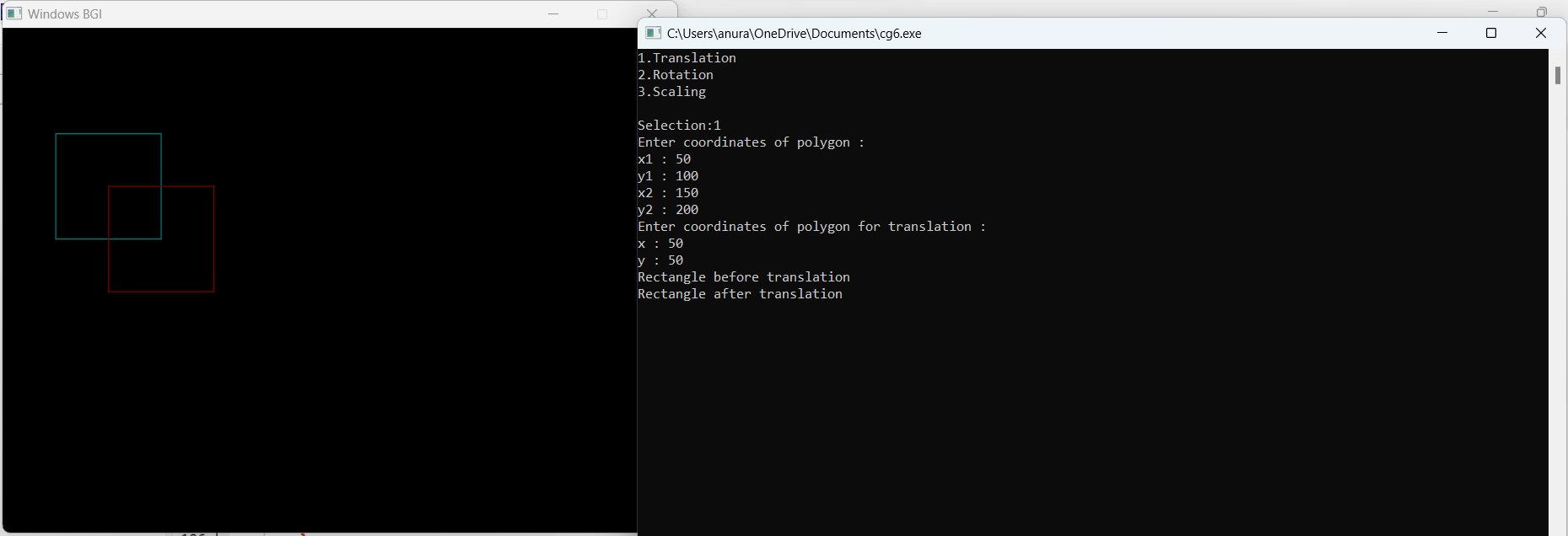
break;

}

}

closegraph();

}



**Question 7**

#include<iostream>

#include<cstdio>

#include<cmath>

using namespace std;

class Matrix3D{

public:

float \*\*arr;

int r,c;

Matrix3D(){}

Matrix3D(int r, int c){

this->r = r;

this->c = c;

this->arr = new float\* [r];

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

arr[i] = new float[c];

}

}

int\* getPlotPoints(){

int \*points = new int[r\*2];

int k=0;

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

points[k++] = (int) arr[i][0];

points[k++] = (int) arr[i][1];

}

points[k++] = (int) arr[0][0];

points[k++] = (int) arr[0][1];

// for(int i =0; i<=2\*r; i+=2){

// cout<<points[i]<<","<<points[i+1]<<"\n";

// }

return points;

}

void setMatrix(float a[][4]){

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

for(int j = 0; j < c; j++){

this->arr[i][j] = a[i][j];

}

}

}

void printMatrix(){

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

cout<<"[ ";

for(int j = 0; j <c; j++){

cout<<arr[i][j]<<" ";

}

i==r-1 ? printf("](%d,%d)\n",r,c): printf("]\n");

}

}

static Matrix3D multiply(Matrix3D m1, Matrix3D m2){

Matrix3D m(m1.r, m2.c);

for(int i = 0; i < m1.r; i++){

for(int j = 0; j < m2.c ; j++){

for(int k = 0; k < m1.c ; k++){

m.arr[i][j] += m1.arr[i][k]\*m2.arr[k][j];

}

}

}

return m;

}

static Matrix3D multiply(Matrix3D\* m1, Matrix3D m2){

Matrix3D m(m1->r, m2.c);

for(int i = 0; i < m1->r; i++){

for(int j = 0; j < m2.c ; j++){

for(int k = 0; k < m1->c ; k++){

m.arr[i][j] += m1->arr[i][k]\*m2.arr[k][j];

}

}

}

return m;

}

};

class Projection{

private:

Matrix3D \*points;

int orthographic(){

cout<<"\nPARALLEL: ORTHOGRAPHIC PROJECTION\*\*\*\n";

Matrix3D trf(4,4);

float t[][4]={

{1,0,0,0},

{0,1,0,0},

{0,0,1,0},

{0,0,0,1},

};

cout<<"Enter projection on:\n";

cout<<"\n1.X=0 plane\n2.Y=0 plane\n3.Z=0 plane\n: ";

int ortho;

cin>>ortho;

if(ortho == 1){

t[0][0] = 0;

}else if(ortho == 2){

t[1][1] = 0;

}else if(ortho == 3){

t[2][2] = 0;

}else{

cout<<"Invalid choice";

return 0;

}

trf.setMatrix(t);

cout<<"\nTransformation matrix:\n";

trf.printMatrix();

cout<<"\nOriginal points:\n";

this->points->printMatrix();

Matrix3D result = Matrix3D::multiply(this->points, trf);

cout<<"\nTransformed points: \n";

result.printMatrix();

return 0;

}

int oblique(){

cout<<"\n\*\*\*PARALLEL: OBLIQUE PROJECTION\*\*\*\n";

cout<<"\*\*\*(Taking horizontal inclination angle = 30 degree)\*\*\*\n";

Matrix3D trf(4,4);

float t[][4]={

{1,0,0,0},

{0,1,0,0},

{0,0,1,0},

{0,0,0,1},

};

cout<<"Enter your choice:\n";

cout<<"\n1.Cavalier projection\n2.Cabinet projection\n: ";

int ob;

cin>>ob;

if(ob == 1){

t[2][0] = -0.866;

t[2][1] = -0.5;

}else if(ob == 2){

t[2][0] = -0.433;

t[2][1] = -0.25;

}else{

cout<<"Invalid choice";

return 0;

}

trf.setMatrix(t);

cout<<"\nTransformation matrix:\n";

trf.printMatrix();

cout<<"\nOriginal points:\n";

this->points->printMatrix();

Matrix3D result = Matrix3D::multiply(this->points, trf);

cout<<"\nTransformed points: \n";

result.printMatrix();

return 0;

}

int axonometric(){

cout<<"\nPARALLEL: AXONOMETRIC PROJECTION\*\*\*\n";

cout<<"\*\*\*(Onto Z=0 plane)\*\*\*\n";

Matrix3D trf(4,4);

float t[][4]={

{1,0,0,0},

{0,1,0,0},

{0,0,1,0},

{0,0,0,1},

};

cout<<"Enter your choice:\n";

cout<<"\n1.Trimetric projection\n2.Dimetric projection\n3.Isometric projection\n: ";

int axono;

cin>>axono;

if(axono == 1){

t[2][2] = 0;

int theta, phi;

cout<<"Enter angle for first rotation: ";

cin>>phi;

cout<<"Enter angle for second rotation: ";

cin>>theta;

float sintheta;

sintheta = sin(theta\*3.14159/180);

t[0][0] = cos(phi\*3.14159/180);

t[2][0] = sin(phi\*3.14159/180);

t[0][1] = sintheta \* t[2][0];

t[1][1] = cos(theta\*3.14159/180);

t[2][1] = -1\*sintheta\*t[0][0];

}else if(axono == 2){

t[2][2] = 0;

float fz,phi,theta,sintheta;

cout<<"Enter foreshortening factor Fz: ";

cin>>fz;

phi = fz / (float) sqrt(2 - fz\*fz);

theta = fz / (float) sqrt(2);

cout<<"Angle for first rotation (phi): "<<phi<<" degree\n";

cout<<"Angle for second rotation (theta): "<<theta<<" degree\n";

sintheta = sin(theta\*3.14159/180);

t[0][0] = cos(phi\*3.14159/180);

t[2][0] = sin(phi\*3.14159/180);

t[0][1] = sintheta \* t[2][0];

t[1][1] = cos(theta\*3.14159/180);

t[2][1] = -1\*sintheta\*t[0][0];

}else if(axono == 3){

t[2][2] = 0;

cout<<"Taking Angle for first rotation (phi): -45 degree\n";

cout<<"Taking Angle for second rotation (theta): 35.26 degree";

t[0][0] = 0.707;

t[2][0] = -0.707;

t[0][1] = -0.408;

t[1][1] = 0.816;

t[2][1] = -0.408;

}else{

cout<<"Invalid choice";

return 0;

}

trf.setMatrix(t);

cout<<"\nTransformation matrix:\n";

trf.printMatrix();

cout<<"\nOriginal points:\n";

this->points->printMatrix();

Matrix3D result = Matrix3D::multiply(this->points, trf);

cout<<"\nTransformed points: \n";

result.printMatrix();

return 0;

}

int parallel(){

cout<<"\n\*\*\*PARALLEL PROJECTION\*\*\*\n";

int choicep1;

cout<<"Enter your choice:\n";

cout<<"\n1.ORTHOGRAPHIC\n2.AXONOMETRIC\n3.OBLIQUE\n: ";

cin>>choicep1;

switch (choicep1)

{

case 1:

orthographic();

break;

case 2:

axonometric();

break;

case 3:

oblique();

break;

default: cout<<"Invalid choice.";

break;

}

return 0;

}

int perspective(){

cout<<"\n\*\*\*PERSPECTIVE PROJECTION\*\*\*\n";

cout<<"\*\*\*(Onto Z=0 plane)\*\*\*\n";

cout<<"\*\*\*(Taking p,q,r to be 0.1 in all 3 cases)\*\*\*\n";

Matrix3D trf(4,4);

float t[][4]={

{1,0,0,0},

{0,1,0,0},

{0,0,1,0},

{0,0,0,1},

};

cout<<"Enter your choice:\n";

cout<<"1.One point projection\n2.Two point projection\n3.Three point projection\n: ";

int pers;

cin>>pers;

if(pers == 1){

t[2][2] = 0;

t[0][3] = 0.1;

}else if(pers == 2){

t[2][2] = 0;

t[0][3] = 0.1;

t[1][3] = 0.1;

}else if(pers == 3){

t[2][2] = 0;

t[0][3] = 0.1;

t[1][3] = 0.1;

t[2][3] = 0.1;

}else{

cout<<"Invalid choice";

return 0;

}

trf.setMatrix(t);

cout<<"\nTransformation matrix:\n";

trf.printMatrix();

cout<<"\nOriginal points:\n";

this->points->printMatrix();

Matrix3D result = Matrix3D::multiply(this->points, trf);

cout<<"\nTransformed points: \n";

result.printMatrix();

return 0;

}

public:

int exec(){

int choice;

while(true){

cout<<"\n\n Select the type of projection:\n";

cout<<"1.Perspective\n2.Parallel\n3.Exit\n:";

cin>>choice;

switch (choice)

{

case 1:

perspective();

break;

case 2:

parallel();

break;

case 3:

return 0;

default:

cout<<"Invalid choice, try again.";

break;

}

}

return 0;

}

Projection(Matrix3D \*t){

this->points = t;

}

};

class transform3d{

public:

Matrix3D \*p;

transform3d(){p = NULL;}

~transform3d(){

}

void getPoints(){

int n;

cout<<"Enter no of points: ";

cin>>n;

p = new Matrix3D(n,4);

float x,y,z;

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

printf("Point %d\nEnter x co-ordinate: ", i+1);

cin>>x;

cout<<"Enter y co-ordinate: ";

cin>>y;

cout<<"Enter z co-ordinate: ";

cin>>z;

p->arr[i][0] = x;

p->arr[i][1] = y;

p->arr[i][2] = z;

p->arr[i][3] = 1;

}

cout<<"Points matrix is:\n";

p->printMatrix();

}

void setPoints(int n,float arr[][4]){

p = new Matrix3D(n,4);

p->setMatrix(arr);

}

void translate(){

float tx, ty,tz;

cout<<"Enter translation in x-axis: ";

cin>>tx;

cout<<"Enter translation in y-axis: ";

cin>>ty;

cout<<"Enter translation in z-axis: ";

cin>>tz;

float t[4][4] = {

{1,0,0,0},

{0,1,0,0},

{0,0,1,0},

{tx,ty,tz,1}

};

Matrix3D trf(4,4);

trf.setMatrix(t);

cout<<"\nTransformation matrix:\n";

trf.printMatrix();

cout<<"\nOriginal points:\n";

this->p->printMatrix();

Matrix3D result = Matrix3D::multiply(this->p, trf);

cout<<"\nTransformed points: \n";

result.printMatrix();

}

void scale(){

float sx, sy,sz;

cout<<"Enter X-axis scaling factor: ";

cin>>sx;

cout<<"Enter Y-axis scaling factor: ";

cin>>sy;

cout<<"Enter Z-axis scaling factor: ";

cin>>sz;

float t[4][4] = {

{sx,0,0,0},

{0,sy,0,0},

{0,0,sz,0},

{0,0,0,1}

};

Matrix3D trf(4,4);

trf.setMatrix(t);

cout<<"\nTransformation matrix:\n";

trf.printMatrix();

cout<<"\nOriginal points:\n";

this->p->printMatrix();

Matrix3D result = Matrix3D::multiply(this->p, trf);

cout<<"\nTransformed points: \n";

result.printMatrix();

}

void rotate(){

float thetaR, thetaD;

cout<<"Enter angle of rotation, in degrees";

cin>>thetaD;

thetaR = thetaD \* 3.14159/180;

float cosT = (float) cos(thetaR);

float sinT = (float) sin(thetaR);

float t[4][4] = {

{1,0,0,0},

{0,1,0,0},

{0,0,1,0},

{0,0,0,1}

};

bool flag = true;

while(flag){

int choice;

cout<<"\nRotate about:\n1.X-Axis\n2.Y-Axis\n3.Z-Axis: ";

cin>>choice;

switch (choice)

{

case 1:

t[1][1] = t[2][2] = cosT;

t[1][2] = -1\*sinT;

t[2][1] = sinT;

flag = false;

break;

case 2:

t[0][0] = t[2][2] = cosT;

t[0][2] = sinT;

t[2][0] = -1\*sinT;

flag = false;

break;

case 3:

t[0][0] = t[1][1] = cosT;

t[0][1] = -1\*sinT;

t[1][0] = sinT;

flag = false;

break;

default: cout<<"Invalid choice, try again";

}

}

Matrix3D trf(4,4);

trf.setMatrix(t);

cout<<"\nTransformation matrix:\n";

trf.printMatrix();

cout<<"\nOriginal points:\n";

this->p->printMatrix();

Matrix3D result = Matrix3D::multiply(this->p, trf);

cout<<"\nTransformed points: \n";

result.printMatrix();

}

void reflect(){

float t[4][4] = {

{1,0,0,0},

{0,1,0,0},

{0,0,1,0},

{0,0,0,1}

};

bool flag = true;

while(flag){

cout<<"\nReflect about:\n1.X-Axis\n2.Y-Axis\n3.Z-Axis: ";

int choice;

cin>>choice;

switch (choice)

{

case 1:

t[1][1] = t[2][2] = -1;

flag = false;

break;

case 2:

t[0][0] = t[2][2] = -1;

flag = false;

break;

case 3:

t[0][0] = t[1][1] = -1;

flag = false;

break;

default: cout<<"Invalid choice, try again";

}

}

Matrix3D trf(4,4);

trf.setMatrix(t);

cout<<"\nTransformation matrix:\n";

trf.printMatrix();

cout<<"\nOriginal points:\n";

this->p->printMatrix();

Matrix3D result = Matrix3D::multiply(this->p, trf);

cout<<"\nTransformed points: \n";

result.printMatrix();

}

};

int main(){

transform3d t;

t.getPoints();

Projection p(t.p);

while(true){

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

cout<<"1.Translate\n2.Scale\n3.Rotate\n4.Reflect\n5.Projection\n6.Show points\n7.Change points\n8.Exit\n: ";

int choice;

cin>>choice;

switch(choice)

{

case 1:

t.translate();

break;

case 2:

t.scale();

break;

case 3:

t.rotate();

break;

case 4:

t.reflect();

break;

case 5:

p.exec();

case 6:

cout<<"\nPoints Matrix is:\n";

t.p->printMatrix();

break;

case 7:

t.getPoints();

break;

case 8:

return 0;

default:

cout<<"\nInvalid choice. Please try again";

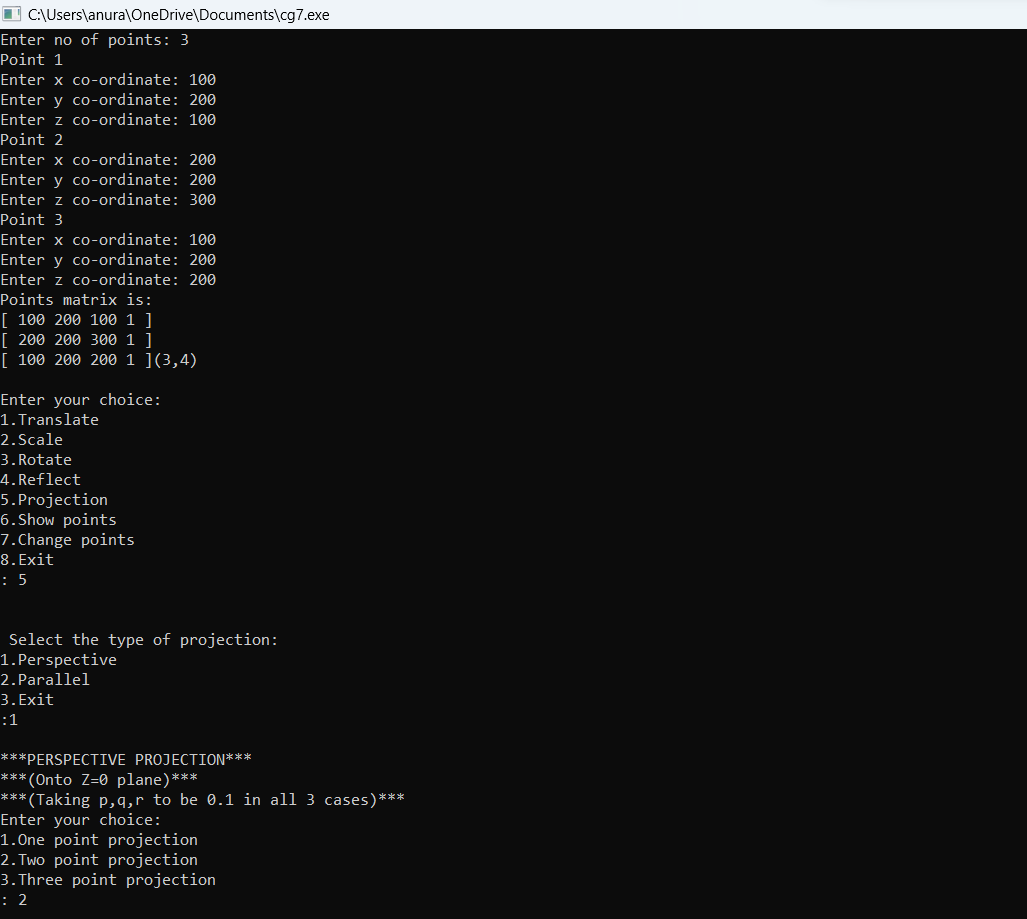
break;

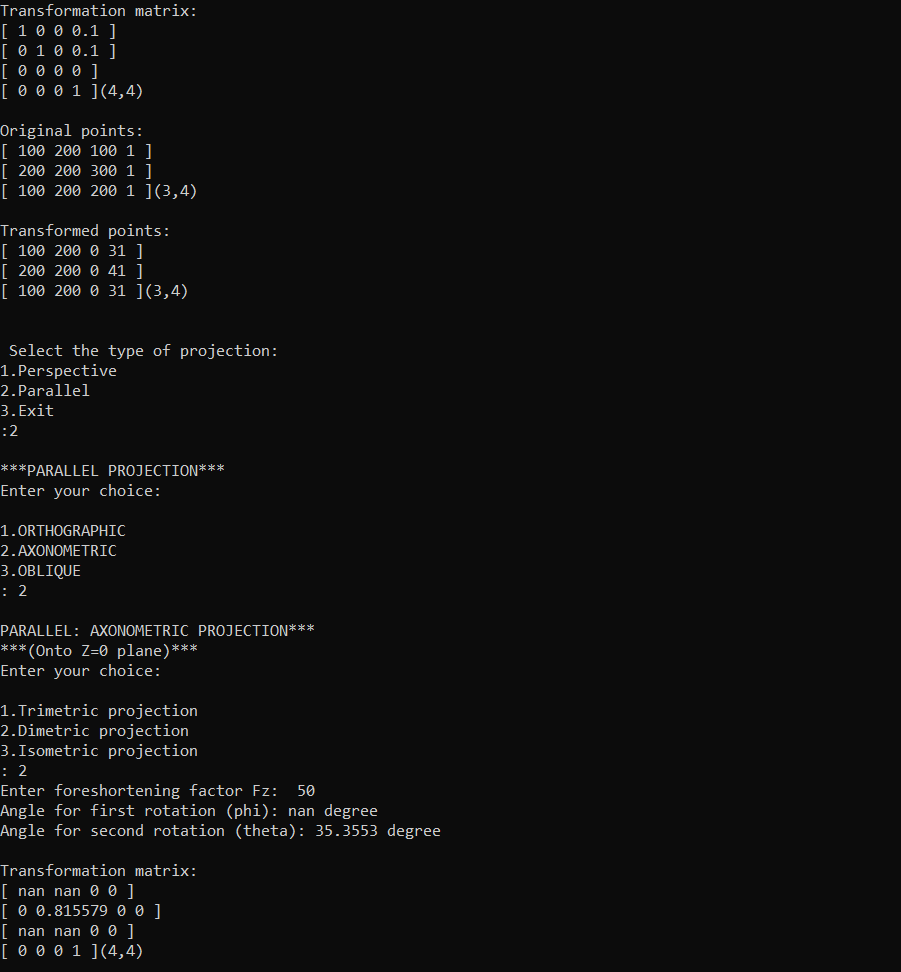
}

}

return 0;

}





**Question 8**

#include<graphics.h>

#include<math.h>

#include <iostream>

using namespace std;

int main()

{

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

double put\_x,put\_y,t;

int gr=DETECT,gm;

initgraph(&gr,&gm," ");

cout<<"\n\*\*\*\*\*\* Bezier Curve \*\*\*\*\*\*\*\*\*\*\*";

cout<<"\n Please enter x and y coordinates ";

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

{

cin>>x[i]>>y[i];

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

}

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

{

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

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

putpixel(put\_x,put\_y, WHITE); // putting pixel

}

getch();

closegraph();

}

