CGL PROGRAMS

/\*

**Title: -** Write C++ program to draw a concave polygon and fill it with desired color using fill algorithm.

**Class:-SE Computer Sub:-**OOPL & CGL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Program- #include<graphics.h> #include<iostream> #include<stdlib.h> using namespace std;

void ffill(int x,int y,int o\_col,int n\_col)

{

int current = getpixel(x,y); if(current==o\_col)

{

delay(1); putpixel(x,y,n\_col); ffill(x+1,y,o\_col,n\_col); ffill(x-1,y,o\_col,n\_col); ffill(x,y+1,o\_col,n\_col); ffill(x,y-1,o\_col,n\_col);

}

}

int main()

{

int x1,y1,x2,y2,x3,y3,xavg,yavg; int gdriver = DETECT,gmode;

initgraph(&gdriver,&gmode,NULL);

cout << " \n\t Enter the points of triangle"; setcolor(1);

cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3; xavg = (int)(x1+x2+x3)/3;

yavg = (int)(y1+y2+y3)/3; line(x1,y1,x2,y2);

line(x2,y2,x3,y3);

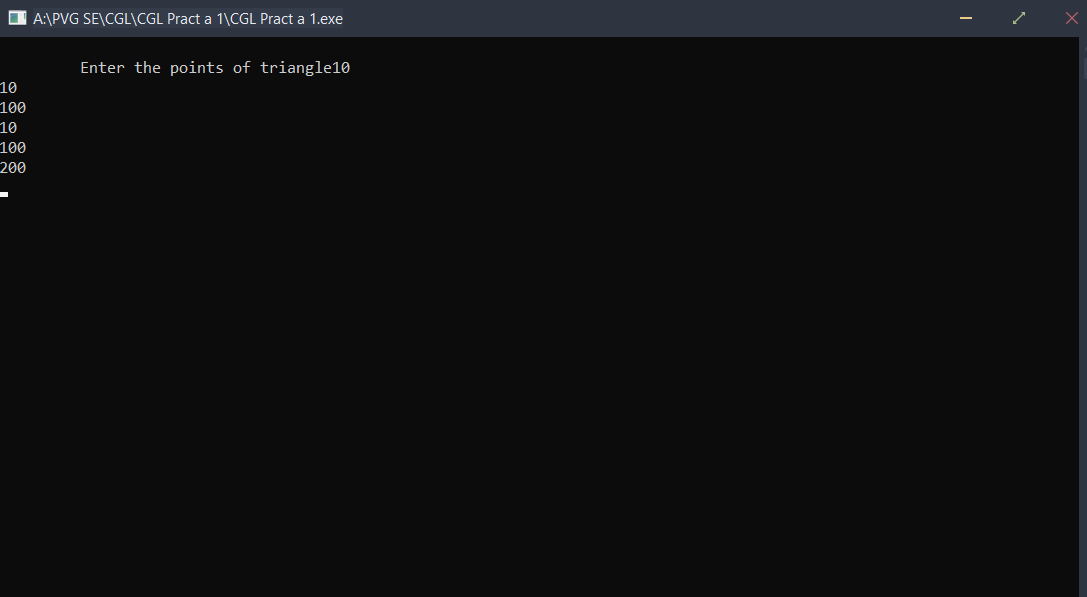
line(x3,y3,x1,y1);

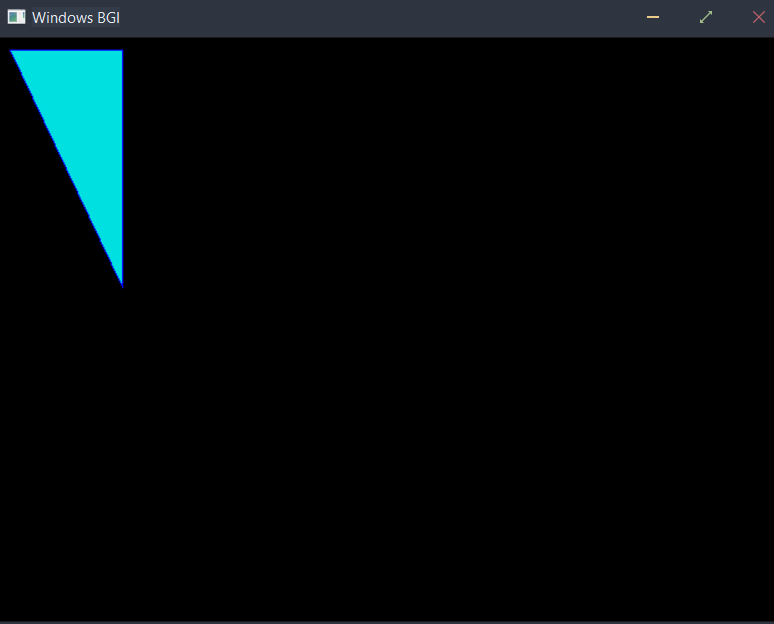
ffill(xavg,yavg,0,3); getch();

return 0;

}

/\*Output:-





/\*

**Title:-** - Write C++ program to implement Cohen Southerland line clipping algorithm.

### Roll No:-

**Class:-SE Computer Sub:-**OOPL & CGL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/ Program- #include<iostream> #include<graphics.h> using namespace std;

static int LEFT=1,RIGHT=2,BOTTOM=4,TOP=8,xl,yl,xh,yh; int getcode(int x,int y)

{

int code = 0;

//Perform Bitwise OR to get outcode if(y > yh) code |=TOP;

if(y < yl) code |=BOTTOM; if(x < xl) code |=LEFT;

if(x > xh) code |=RIGHT; return code;

}

int main()

{

int gdriver = DETECT,gmode; initgraph(&gdriver,&gmode,NULL); setcolor(BLUE);

cout<<"Enter bottom left and top right co-ordinates of window: "; cin>>xl>>yl>>xh>>yh;

rectangle(xl,yl,xh,yh); int x1,y1,x2,y2;

cout<<"Enter the endpoints of the line: "; cin>>x1>>y1>>x2>>y2; line(x1,y1,x2,y2);

getch();

int outcode1=getcode(x1,y1), outcode2=getcode(x2,y2); int accept = 0; //decides if line is to be drawn

while(1)

{

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

//Both points inside. Accept line

if(outcode1==0 && outcode2==0)

{

accept = 1; break;

}

//AND of both codes != 0.Line is outside. Reject line else if((outcode1 & outcode2)!=0)

{

}

else

{

break;

int x,y; int temp;

//Decide if point1 is inside, if not, calculate intersection if(outcode1==0)

temp = outcode2; else

temp = outcode1;

//Line clips top edge if(temp & TOP)

{

x = x1+ (yh-y1)/m; y = yh;

}

else if(temp & BOTTOM)

{

//Line clips bottom edge x = x1+ (yl-y1)/m;

y = yl;

}

else if(temp & LEFT)

{

//Line clips left edge x = xl;

y = y1+ m\*(xl-x1);

}

else if(temp & RIGHT)

{

//Line clips right edge x = xh;

y = y1+ m\*(xh-x1);

}

//Check which point we had selected earlier as temp, and replace its coordinates if(temp == outcode1)

{

}

else

{

}

}

}

x1 = x; y1 = y;

outcode1 = getcode(x1,y1);

x2 = x; y2 = y;

outcode2 = getcode(x2,y2);

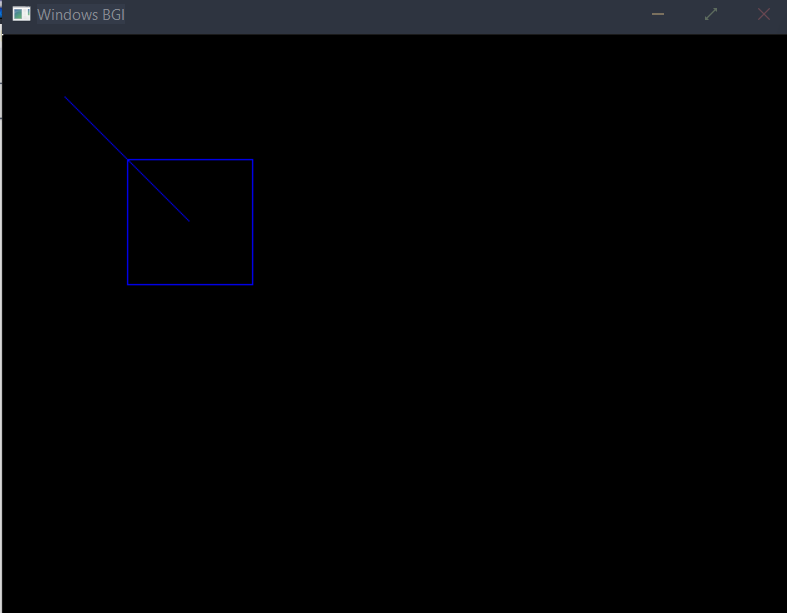
setcolor(WHITE); cout<<"After clipping:"; if(accept) line(x1,y1,x2,y2); delay(8000);

return 0; closegraph();

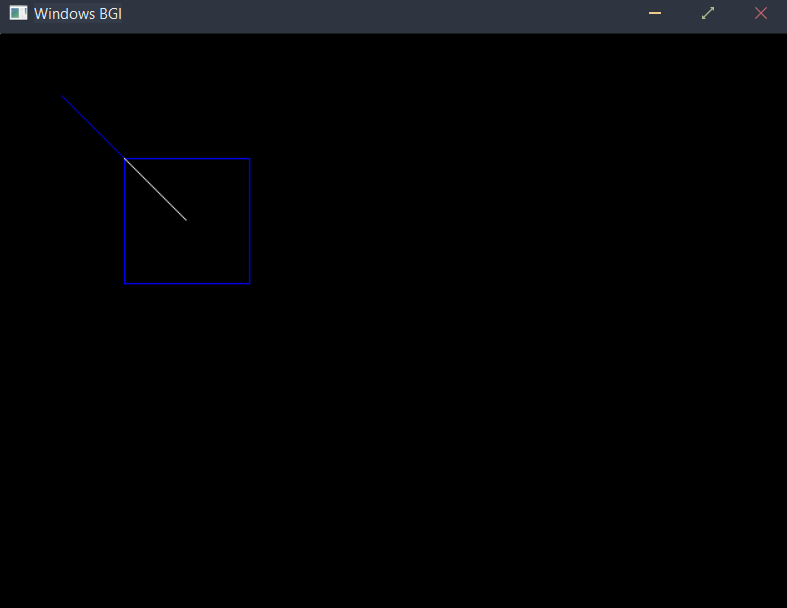
}

## /\*Output:-

**Before Clipping :-**



**After clipping :-**



/\*

CGL Assignment No 3

**Title:-** Write C++ program to draw the following pattern. Use DDA line and Bresenham drawing algorithm. Apply the concept of encapsulation.

### Roll No:-

**Class:-SE Computer Sub:-**OOPL & CGL

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*/

**Program-** #include<iostream> #include<graphics.h> #include <bits/stdc++.h> using namespace std; class algo

{

public:

void dda\_line(float x1, float y1, float x2, float y2); void bresneham\_cir(int r);

};

void algo::dda\_line(float x1, float y1, float x2, float y2)

{

float x,y,dx,dy,step; int i;

//step 2 dx=abs(x2-x1); dy=abs(y2-y1);

cout<<"dy="<<dy<<"\tdx="<<dx;

//step 3 if(dx>=dy) step=dx; else step=dy;

cout<<"\n"<<step<<endl;

//step 4

float xinc=float((x2-x1)/step); float yinc=float((y2-y1)/step);

//step 5 x=x1; y=y1;

// outtextxy(0,0,"(0,0)");

//step 6 i=1;

while(i<=step)

{

//cout<<endl<<"\t"<<i<<"\t(x,y)=("<<x<<","<<y<<")"; putpixel(320+x,240-y,4);

x=x+xinc; y=y+yinc; i=i+1;

// delay(10);

}

}

void algo::bresneham\_cir(int r)

{

float x,y,p; x=0;

y=r;

p=3-(2\*r); while(x<=y)

{

putpixel(320+x,240+y,1); putpixel(320-x,240+y,2); putpixel(320+x,240-y,3); putpixel(320-x,240-y,5); putpixel(320+y,240+x,6); putpixel(320+y,240-x,7); putpixel(320-y,240+x,8); putpixel(320-y,240-x,9); x=x+1;

if(p<0)

{

}

else

{

}

p=p+4\*(x)+6;

p=p+4\*(x-y)+10; y=y-1;

}

}

int main()

// delay(20);

{

algo a1; int i;

float r,ang,r1; initwindow(630,480); cout<<"Enter radius of circle"; cin>>r; a1.bresneham\_cir((int)r); ang=3.24/180;

float c=r\*cos(30\*ang); float s=r\*sin(30\*ang); a1.dda\_line(0,r,0-c,0-s);

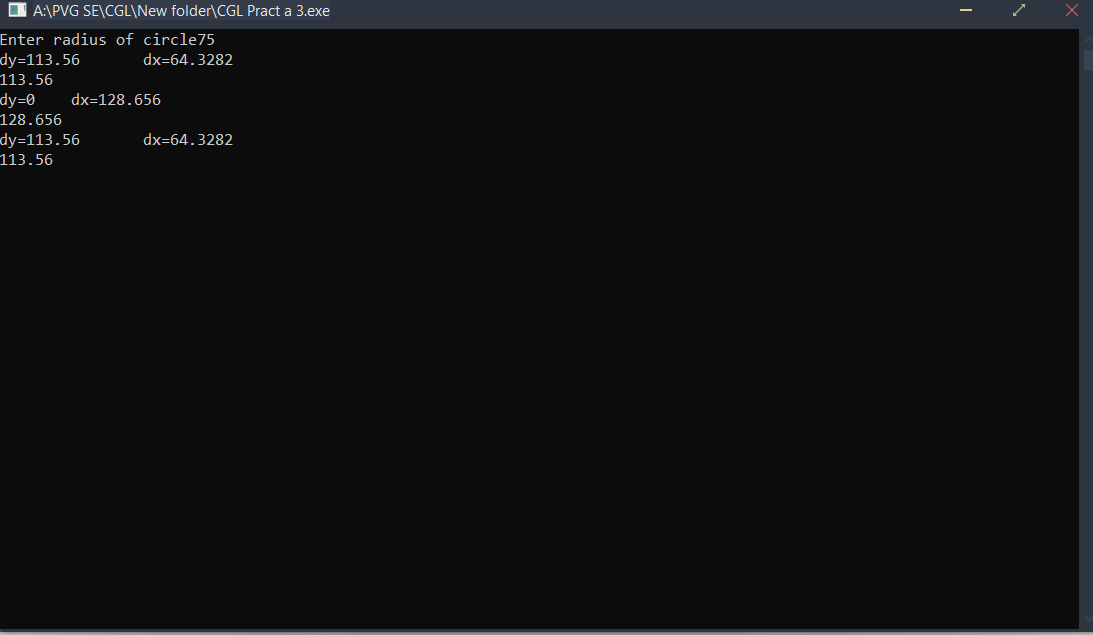
a1.dda\_line(0-c,0-s,0+c,0-s);

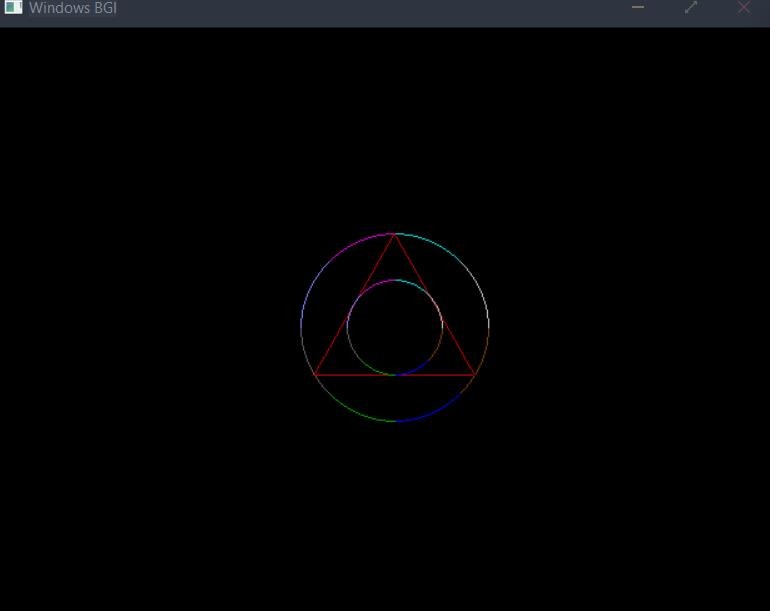
a1.dda\_line(0+c,0-s,0,r); r1=s; a1.bresneham\_cir((int)r1); getch();

closegraph(); return 0;

}

## /\*Output:-





/\*

**Title: -** Write C++ program to draw 2-D object and perform following basic transformations, Scaling b) Translation c) Rotation. Apply the concept of operator overloading.

### Roll No:-

**Class:-SE Computer Sub:-**OOPL & CGL

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*/

**Program-** #include<iostream> #include<stdlib.h> #include<graphics.h> #include<math.h> using namespace std; class POLYGON

{

private:

int p[10][10],Trans\_result[10][10],Trans\_matrix[10][10]; float Rotation\_result[10][10],Rotation\_matrix[10][10]; float Scaling\_result[10][10],Scaling\_matrix[10][10]; float Shearing\_result[10][10],Shearing\_matrix[10][10]; int Reflection\_result[10][10],Reflection\_matrix[10][10]; public:

int accept\_poly(int [][10]); void draw\_poly(int [][10],int);

void draw\_polyfloat(float [][10],int);

void matmult(int [][10],int [][10],int,int,int,int [][10]);

void matmultfloat(float [][10],int [][10],int,int,int,float [][10]); void shearing(int [][10],int);

void scaling(int [][10],int); void rotation(int [][10],int); void translation(int [][10],int); void reflection(int [][10],int);

};

int POLYGON :: accept\_poly(int p[][10])

{

int i,n;

cout<<"\n\n\t\tEnter no.of vertices:"; cin>>n;

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

{

cout<<"\n\n\t\tEnter (x,y)Co-ordinate of point P"<<i<<": "; cin >> p[i][0] >> p[i][1];

p[i][2] = 1;

}

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

{

cout<<"\n";

for(int j=0;j<3;j++)

{

cout<<p[i][j]<<"\t";

}

}

return n;

}

void POLYGON :: draw\_poly(int p[][10], int n)

{

int i,gd = DETECT,gm; initgraph(&gd,&gm,NULL); line(320,0,320,480); line(0,240,640,240);

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

{

if(i<n-1)

{

}

else

}

line(p[i][0]+320, -p[i][1]+240, p[i+1][0]+320, -p[i+1][1]+240);

line(p[i][0]+320, -p[i][1]+240, p[0][0]+320, -p[0][1]+240);

delay(3000);

}

void POLYGON :: draw\_polyfloat(float p[][10], int n)

{

int i,gd = DETECT,gm; initgraph(&gd,&gm,NULL); line(320,0,320,480); line(0,240,640,240);

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

{

if(i<n-1)

{

line(p[i][0]+320, -p[i][1]+240, p[i+1][0]+320, -p[i+1][1]+240);

}

else

}

line(p[i][0]+320, -p[i][1]+240, p[0][0]+320, -p[0][1]+240);

//delay(8000);

}

void POLYGON :: translation(int p[10][10],int n)

{

int tx,ty,i,j; int i1,j1,k1,r1,c1,c2; r1=n;c1=c2=3;

cout << "\n\n\t\tEnter X-Translation tx: "; cin >> tx;

cout << "\n\n\t\tEnter Y-Translation ty: "; cin >> ty;

for(i=0;i<3;i++) for(j=0;j<3;j++) Trans\_matrix[i][j] = 0;

Trans\_matrix[0][0] = Trans\_matrix[1][1] = Trans\_matrix[2][2] = 1; Trans\_matrix[2][0] = tx;

Trans\_matrix[2][1] = ty; for(i1=0;i1<10;i1++) for(j1=0;j1<10;j1++) Trans\_result[i1][j1] = 0; for(i1=0;i1<r1;i1++) for(j1=0;j1<c2;j1++) for(k1=0;k1<c1;k1++)

Trans\_result[i1][j1] = Trans\_result[i1][j1]+(p[i1][k1] \* Trans\_matrix[k1][j1]); cout << "\n\n\t\tPolygon after Translationâ€¦";

draw\_poly(Trans\_result,n);

}

void POLYGON :: rotation(int p[][10],int n)

{

float type,Ang,Sinang,Cosang; int i,j; int i1,j1,k1,r1,c1,c2; r1=n;c1=c2=3;

cout << "\n\n\t\tEnter the angle of rotation in degrees: "; cin >> Ang;

cout << "\n\n \*\*\*\* Rotation Types \*\*\*\*";

cout << "\n\n\t\t1.Clockwise Rotation \n\n\t\t2.Anti-Clockwise Rotation "; cout << "\n\n\t\tEnter your choice(1-2): ";

cin >> type;

Ang = (Ang \* 6.2832)/360;

Sinang = sin(Ang);

Cosang = cos(Ang); cout<<"Mark1"; for(i=0;i<3;i++) for(j=0;j<3;j++) Rotation\_matrix[i][j] = 0; cout<<"Mark2";

Rotation\_matrix[0][0] = Rotation\_matrix[1][1] = Cosang; Rotation\_matrix[0][1] = Rotation\_matrix[1][0] = Sinang; Rotation\_matrix[2][2] = 1;

if(type == 1) Rotation\_matrix[0][1] = -Sinang; else

Rotation\_matrix[1][0] = -Sinang; for(i1=0;i1<10;i1++) for(j1=0;j1<10;j1++) Rotation\_result[i1][j1] = 0; for(i1=0;i1<r1;i1++) for(j1=0;j1<c2;j1++) for(k1=0;k1<c1;k1++)

Rotation\_result[i1][j1] = Rotation\_result[i1][j1]+(p[i1][k1] \* Rotation\_matrix[k1][j1]);

cout << "\n\n\t\tPolygon after Rotationâ€¦"; for(i=0;i<n;i++)

{

cout<<"\n";

for(int j=0;j<3;j++)

{

cout<<Rotation\_result[i][j]<<"\t";

}

}

draw\_polyfloat(Rotation\_result,n);

}

void POLYGON :: scaling(int p[][10],int n)

{

float Sx,Sy;

int i,j; int i1,j1,k1,r1,c1,c2; r1=n;c1=c2=3;

cout<<"\n\n\t\tEnter X-Scaling Sx: "; cin>>Sx;

cout<<"\n\n\t\tEnter Y-Scaling Sy: "; cin>>Sy;

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

{

for(j=0;j<3;j++)

{

Scaling\_matrix[i][j] = 0;

}

}

Scaling\_matrix[0][0] = Sx; Scaling\_matrix[0][1] = 0;

Scaling\_matrix[0][2] = 0;

Scaling\_matrix[1][0] = 0; Scaling\_matrix[1][1] = Sy; Scaling\_matrix[1][2] = 0;

Scaling\_matrix[2][0] = 0;

Scaling\_matrix[2][1] = 0;

Scaling\_matrix[2][2] = 1; for(i1=0;i1<10;i1++) for(j1=0;j1<10;j1++) Scaling\_result[i1][j1] = 0; for(i1=0;i1<r1;i1++) for(j1=0;j1<c2;j1++) for(k1=0;k1<c1;k1++)

Scaling\_result[i1][j1] = Scaling\_result[i1][j1]+(p[i1][k1] \* Scaling\_matrix[k1][j1]);

cout<<"\n\n\t\tPolygon after Scalingâ€¦"; draw\_polyfloat(Scaling\_result,n);

}

int main()

{

int ch,n,p[10][10]; POLYGON p1;

cout<<"\n\n \*\*\*\* 2-D TRANSFORMATION \*\*\*\*"; n= p1.accept\_poly(p);

cout <<"\n\n\t\tOriginal Polygon â€¦"; p1.draw\_poly(p,n);

do

{

int ch;

cout<<"\n\n \*\*\*\* 2-D TRANSFORMATION \*\*\*\*"; cout<<"\n\n\t\t1.Translation \n\n\t\t2.Scaling \n\n\t\t3.Rotation \n\n\t\t4.Exit"; cout<<"\n\n\tEnter your choice(1-6):";

cin>>ch; switch(ch)

{

case 1:

//cout<<"case1"; p1.translation(p,n); break;

case 2: cout<<"case2"; p1.scaling(p,n); break;

case 3: cout<<"case3"; p1.rotation(p,n); break;

case 4: exit(0);

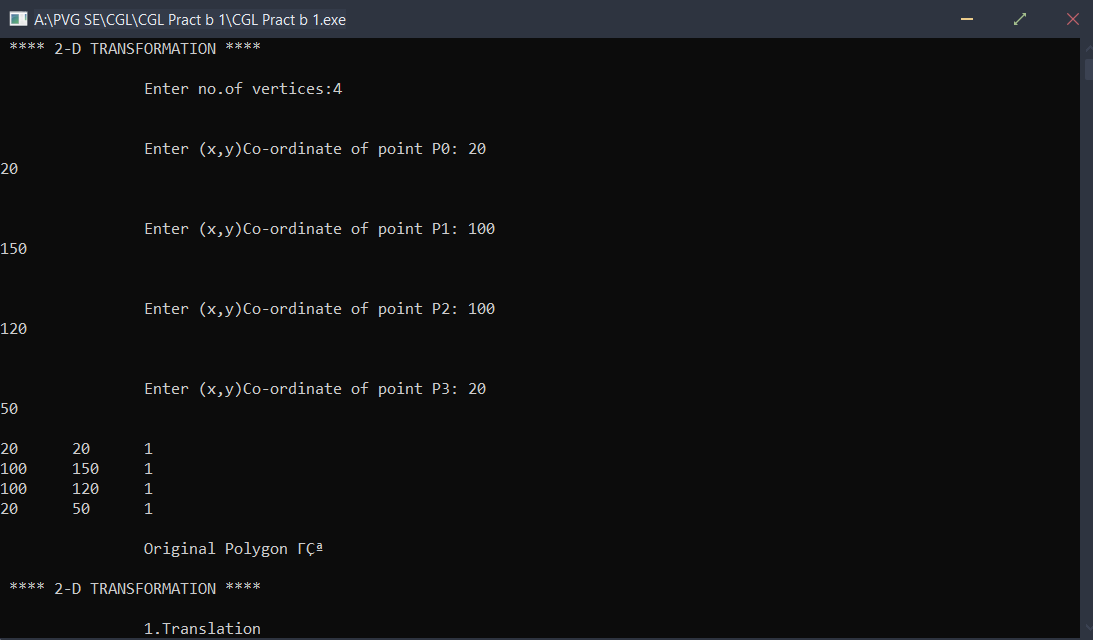
}

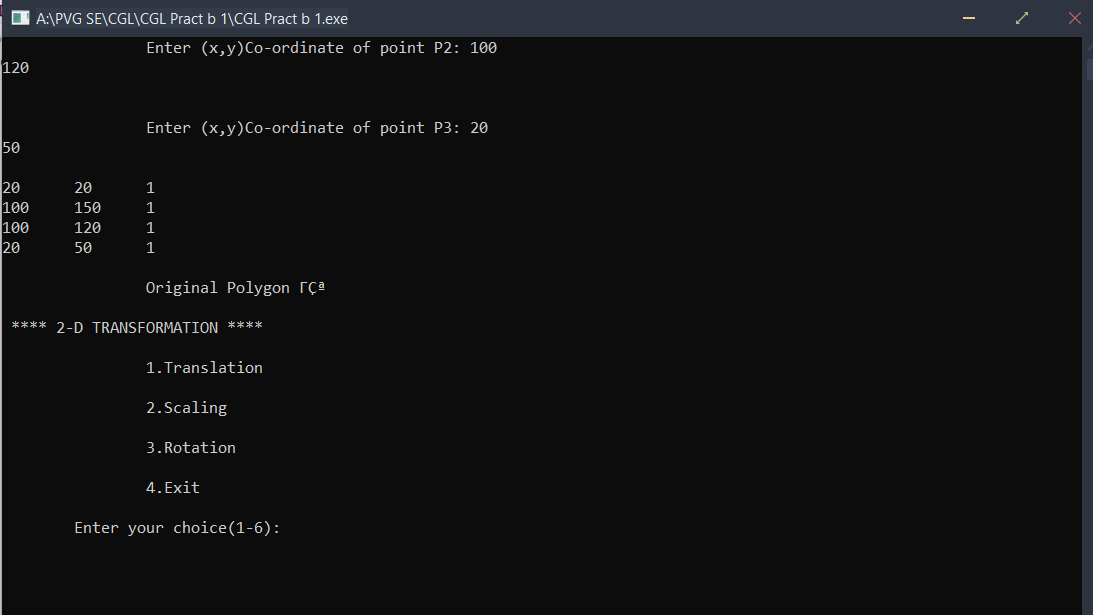
}

while(1); return 0;

}

/\*Output:

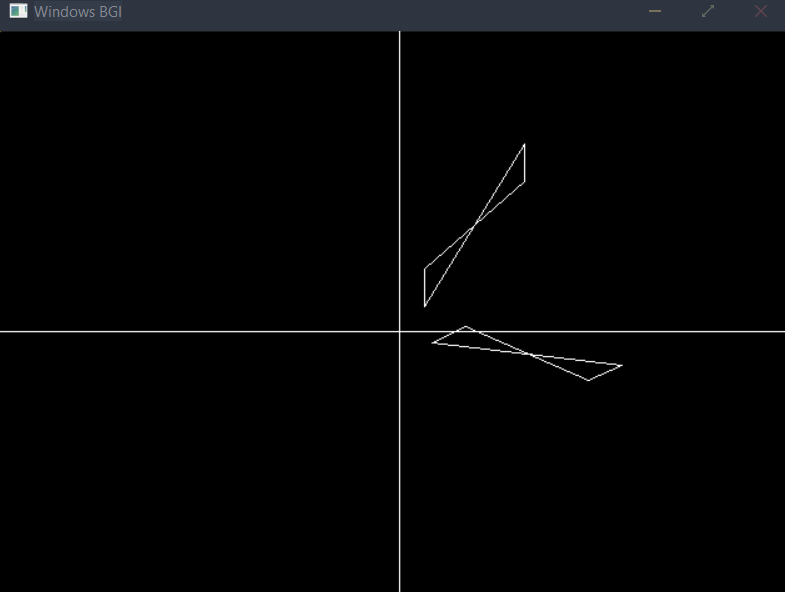




# After Translation:- Tx=20, Ty=20



**After Rotation at 65 degree:-**



\*/

/\*

**Title: -** Write C++ program to generate Hilbert curve using concept of fractals.

### Roll No:-

**Class:-SE Computer Sub:-**OOPL & CGL

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*/

**Program-** #include<iostream> #include<graphics.h> #include<math.h> #include<cstdlib> using namespace std;

void move(int j, int h, int &x,int &y)

{

if(j==1) y-=h; else

if(j==2)

x+=h; else if(j==3)

y+=h; else if(j==4)

x-=h;

lineto(x,y);

}

void hilbert(int r,int d,int l ,int u,int i,int h,int &x,int &y)

{

if(i>0)

{

i--;

hilbert(d,r,u,l,i,h,x,y); move(r,h,x,y); hilbert(r,d,l,u,i,h,x,y); move(d,h,x,y); hilbert(r,d,l,u,i,h,x,y); move(l,h,x,y); hilbert(u,l,d,r,i,h,x,y);

}

}

int main()

{

int n,x1,y1;

int x0=50,y0=150,x,y,h=10,r=2,d=3,l=4,u=1;

cout<<"Give the value of n="; cin>>n;

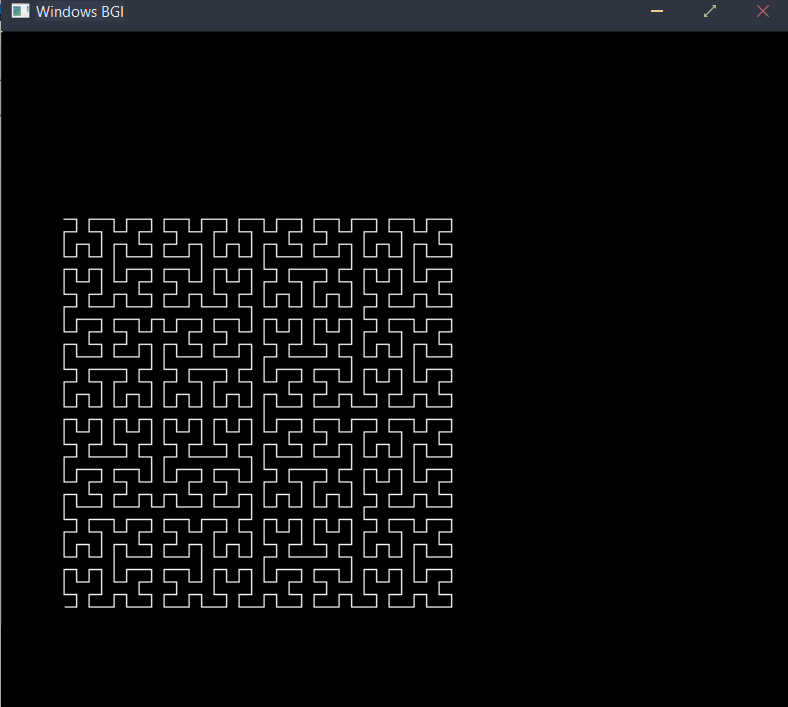
x=x0; y=y0;

int driver=DETECT,mode=0; initgraph(&driver,&mode,NULL); moveto(x,y); hilbert(r,d,l,u,n,h,x,y); delay(10000);

closegraph(); return 0;

}

# /\*Output:-



/\*

**Title: -** Write OpenGL Program to draw Sunrise and Sun Set.

### Roll No:-

**Class:-SE Computer Sub:-**OOPL & CGL

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*/

**Program-** #include<graphics.h> int main()

{

int gd = DETECT, gm; initgraph(&gd, &gm,NULL); int midx,midy,r=10; midx=getmaxx()/2; while(r<=50)

{

cleardevice(); setcolor(WHITE); line(0,310,160,150); line(160,150,320,310); line(320,310,480,150); line(480,150,640,310); line(0,310,640,310);

arc(midx,310,225,133,r); floodfill(midx,300,15); if(r>20)

{

setcolor(7); floodfill(2,2,15); setcolor(6); floodfill(150,250,15); floodfill(550,250,15); setcolor(2); floodfill(2,450,15);

}

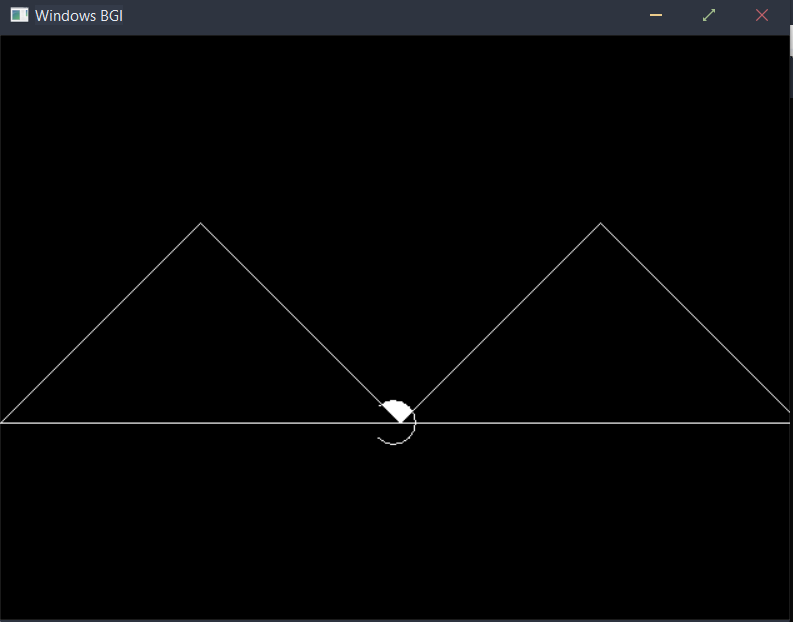
delay(1000); r+=2;

}

getch(); closegraph();

}

# /\*Output:-



/\*

**Title: -** Write C++ program to generate Bouncing ball. Apply the concept of polymorphism.

### Roll No:-

**Class:-SE Computer Sub:-**OOPL & CGL

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*/

**Program-** #include <iostream> #include <cstdlib>

#include <graphics.h> using namespace std; int main()

{

int gd = DETECT, gm; int i, x, y, flag=0;

initgraph(&gd, &gm,NULL);

/\* get mid positions in x and y-axis \*/ x = getmaxx()/2;

y = 30;

while (1)

{

if(y >= getmaxy()-30 || y <= 30) flag = !flag;

/\* draws the gray board \*/ setcolor(RED);

//setfillstyle(SOLID\_FILL, RED); circle(x, y, 30);

floodfill(x, y, RED);

/\* delay for 50 milli seconds \*/ delay(50);

/\* clears screen \*/ cleardevice(); if(flag)

{

y = y + 5;

}

else

{

y = y - 5;

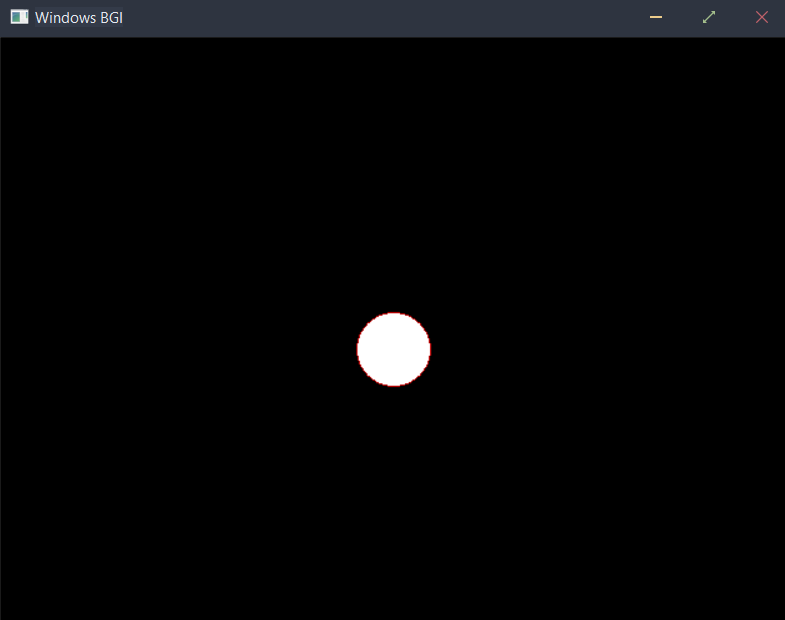
}

}

delay(5000); closegraph(); return 0;

}

# /\*Output:-



/\*

# Mini Project

**Title: -** Write a C++ program to draw a man walking in rain with an umbrella.

### Roll No:-

**Class:-SE Computer Sub:-**OOPL & CGL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Program- #include<stdio.h> #include<graphics.h>

#define ScreenWidth getmaxx() #define ScreenHeight getmaxy() #define GroundY ScreenHeight\*0.80 int ldisp=0;

void DrawManAndUmbrella(int x,int ldisp)

{

//Man's head circle(x,GroundY-90,10);

line(x,GroundY-80,x,GroundY-30);

//Man's hand

line(x,GroundY-70,x+10,GroundY-60); line(x,GroundY-65,x+10,GroundY-55); line(x+10,GroundY-60,x+20,GroundY-70); line(x+10,GroundY-55,x+20,GroundY-70);

//Man's legs

line(x,GroundY-30,x+ldisp,GroundY); line(x,GroundY-30,x-ldisp,GroundY);

//umbrella

pieslice(x+20,GroundY-120,0,180,40); line(x+20,GroundY-120,x+20,GroundY-70);

}

void Rain(int x)

{

int i,rx,ry; for(i=0;i<400;i++)

{

rx=rand() % ScreenWidth;

ry=rand() % ScreenHeight; if(ry<GroundY-4)

{

if(ry<GroundY-120 || (ry>GroundY-120 && (rx<x-20 || rx>x+60))) line(rx,ry,rx+0.5,ry+4);

}

}

}

int main()

{

int gd=DETECT,gm,x=0; initgraph(&gd,&gm,NULL);

while(!kbhit())

{

//Draw Ground line(0,GroundY,ScreenWidth,GroundY); Rain(x);

ldisp=(ldisp+2)%20; DrawManAndUmbrella(x,ldisp);

delay(75); cleardevice(); x=(x+2)%ScreenWidth;

}

getch();

}

/\*Output:-

