**Name** – Tejas Pravin Pawar ( SEB25 )

**Assignment no – 6**

**Title** - Write an OpenGL program to Sunrise and Sunset.

**Source Code :-**

#include<iostream>

#include<stdlib.h>

#ifdef \_\_APPLE\_\_

#include<openGL/openGL.h>

#include<GLUT/glut.h>

#else

#include<GL/glut.h>

#endif

using namespace std;

float ballX = -0.8f;

float ballY = -0.3f;

float ballZ = -1.2f;

float colR=3.0;

float colG=1.5;

float colB=1.0;

float bgColR=0.0;

float bgColG=0.0;

float bgColB=0.0;

static int flag=1;

void drawBall(void) {

glColor3f(colR,colG,colB); //set ball colour

glTranslatef(ballX,ballY,ballZ); //moving it toward the screen a bit

on creation

glutSolidSphere (0.05, 30, 30); //create ball.

}

void drawAv(void) {

glBegin(GL\_POLYGON);

glColor3f(1.0,1.0,1.0);

glVertex3f(-0.9,-0.7,-1.0);

glVertex3f(-0.5,-0.1,-1.0);

glVertex3f(-0.2,-1.0,-1.0);

glVertex3f(0.5,0.0,-1.0);

glVertex3f(0.6,-0.2,-1.0);

glVertex3f(0.9,-0.7,-1.0);

glEnd();

}

void drawClouds(){}

void keyPress(int key, int x, int y)

{

if(key==GLUT\_KEY\_RIGHT)

ballX -= 0.05f;

if(key==GLUT\_KEY\_LEFT)

ballX += 0.05f;

glutPostRedisplay();

}

void initRendering() {

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_LIGHTING); //Enable lighting

glEnable(GL\_LIGHT0); //Enable light #0

glEnable(GL\_LIGHT1); //Enable light #1

glEnable(GL\_NORMALIZE); //Automatically normalize normals

//glShadeModel(GL\_SMOOTH); //Enable smooth shading

}

//Called when the window is resized

void handleResize(int w, int h) {

//Tell OpenGL how to convert from coordinates to pixel values

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION); //Switch to setting the camera perspective

//Set the camera perspective

glLoadIdentity(); //Reset the camera

gluPerspective(45.0, //The camera angle

(double)w / (double)h, //The width-to-height ratio

1.0, //The near z clipping coordinate

200.0); //The far z clipping coordinate

}

void drawScene()

{

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

glClearColor(bgColR,bgColG,bgColB,0.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

//Add ambient light

GLfloat ambientColor[] = {0.2f, 0.2f, 0.2f, 1.0f}; //Color (0.2, 0.2,

0.2)

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, ambientColor);

//Add positioned light

GLfloat lightColor0[] = {0.5f, 0.5f, 0.5f, 1.0f}; //Color (0.5, 0.5, 0.5)

GLfloat lightPos0[] = {4.0f, 0.0f, 8.0f, 1.0f}; //Positioned at (4, 0, 8)

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightColor0);

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPos0);

//Add directed light

GLfloat lightColor1[] = {0.5f, 0.2f, 0.2f, 1.0f}; //Color (0.5, 0.2, 0.2)

//Coming from the direction (-1, 0.5, 0.5)

GLfloat lightPos1[] = {-1.0f, 0.5f, 0.5f, 0.0f};

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, lightColor1);

glLightfv(GL\_LIGHT1, GL\_POSITION, lightPos1);

//drawing the SUN

glPushMatrix();

drawBall();

glPopMatrix();

//drawing the Mount Avarest

glPushMatrix();

drawAv();

glPopMatrix();

//drawing the Clouds

glPushMatrix();

drawClouds();

glPopMatrix();

glutSwapBuffers();

}

//float \_angle = 30.0f;

void update(int value) {

if(ballX>0.9f)

{

ballX = -0.8f;

ballY = -0.3f;

flag=1;

colR=2.0;

colG=1.50;

colB=1.0;

bgColB=0.0;

}

if(flag)

{

ballX += 0.001f;

ballY +=0.0007f;

colR-=0.001;

//colG+=0.002;

colB+=0.005;

bgColB+=0.001;

if(ballX>0.01)

{

flag=0;

}

}

if (!flag)

{

ballX += 0.001f;

ballY -=0.0007f;

colR+=0.001;

colB-=0.01;

bgColB-=0.001;

if(ballX<-0.3)

{

flag=1;

}

}

glutPostRedisplay(); //Tell GLUT that the display has changed

//Tell GLUT to call update again in 25 milliseconds

glutTimerFunc(25, update, 0);

}

int main(int argc,char\*\* argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB|GLUT\_DEPTH);

glutInitWindowSize(400,400);

glutCreateWindow("Sun");

initRendering();

glutDisplayFunc(drawScene);

glutFullScreen();

glutSpecialFunc(keyPress);

glutReshapeFunc(handleResize);

glutTimerFunc(25, update, 0);

glutMainLoop();

return(0);

}

Output :-

1. For Sunrise:-

if(key==GLUT\_KEY\_LEFT)

ballX += 0.05f;

glutPostRedisplay();

}

void initRendering() {

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_LIGHTING); //Enable lighting

glEnable(GL\_LIGHT0); //Enable light #0

glEnable(GL\_LIGHT1); //Enable light #1

glEnable(GL\_NORMALIZE); //Automatically normalize normals

//glShadeModel(GL\_SMOOTH); //Enable smooth shading

}

//Called when the window is resized

void handleResize(int w, int h) {

//Tell OpenGL how to convert from coordinates to pixel values

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION); //Switch to setting the camera perspective

//Set the camera perspective

glLoadIdentity(); //Reset the camera

gluPerspective(45.0, //The camera angle

(double)w / (double)h, //The width-to-height ratio

1.0, //The near z clipping coordinate

200.0); //The far z clipping coordinate

}

void drawScene()

{

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

glClearColor(bgColR,bgColG,bgColB,0.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

//Add ambient light

GLfloat ambientColor[] = {0.2f, 0.2f, 0.2f, 1.0f}; //Color (0.2, 0.2,

0.2)

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, ambientColor);

//Add positioned light

GLfloat lightColor0[] = {0.5f, 0.5f, 0.5f, 1.0f}; //Color (0.5, 0.5, 0.5)

GLfloat lightPos0[] = {4.0f, 0.0f, 8.0f, 1.0f}; //Positioned at (4, 0, 8)

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightColor0);

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPos0);

//Add directed light

GLfloat lightColor1[] = {0.5f, 0.2f, 0.2f, 1.0f}; //Color (0.5, 0.2, 0.2)

//Coming from the direction (-1, 0.5, 0.5)

GLfloat lightPos1[] = {-1.0f, 0.5f, 0.5f, 0.0f};

glLightfv(GL\_LIGHT1, GL\_DIFFUSE, lightColor1);

glLightfv(GL\_LIGHT1, GL\_POSITION, lightPos1);

//drawing the SUN

glPushMatrix();

drawBall();

glPopMatrix();

//drawing the Mount Avarest

glPushMatrix();

drawAv();

glPopMatrix();

//drawing the Clouds

glPushMatrix();

drawClouds();

glPopMatrix();

glutSwapBuffers();

}

//float \_angle = 30.0f;

void update(int value) {

if(ballX>0.9f)

{

ballX = -0.8f;

ballY = -0.3f;

flag=1;

colR=2.0;

colG=1.50;

colB=1.0;

bgColB=0.0;

}

if(flag)

{

ballX += 0.001f;

ballY +=0.0007f;

colR-=0.001;

//colG+=0.002;

colB+=0.005;

bgColB+=0.001;

if(ballX>0.01)

{

flag=0;

}

}

if (!flag)

{

ballX += 0.001f;

ballY -=0.0007f;

colR+=0.001;

colB-=0.01;

bgColB-=0.001;

if(ballX<-0.3)

{

flag=1;

}

}

glutPostRedisplay(); //Tell GLUT that the display has changed

//Tell GLUT to call update again in 25 milliseconds

glutTimerFunc(25, update, 0);

}

int main(int argc,char\*\* argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB|GLUT\_DEPTH);

glutInitWindowSize(400,400);

glutCreateWindow("Sun");

initRendering();

glutDisplayFunc(drawScene);

glutFullScreen();

glutSpecialFunc(keyPress);

glutReshapeFunc(handleResize);

glutTimerFunc(25, update, 0);

glutMainLoop();

return(0);

}

Output :-

1. For Sunrise:-

//drawing the Mount Avarest

glPushMatrix();

drawAv();

glPopMatrix();

//drawing the Clouds

glPushMatrix();

drawClouds();

glPopMatrix();

glutSwapBuffers();

}

//float \_angle = 30.0f;

void update(int value) {

if(ballX>0.9f)

{

ballX = -0.8f;

ballY = -0.3f;

flag=1;

colR=2.0;

colG=1.50;

colB=1.0;

bgColB=0.0;

}

if(flag)

{

ballX += 0.001f;

ballY +=0.0007f;

colR-=0.001;

//colG+=0.002;

colB+=0.005;

bgColB+=0.001;

if(ballX>0.01)

{

flag=0;

}

}

if (!flag)

{

ballX += 0.001f;

ballY -=0.0007f;

colR+=0.001;

colB-=0.01;

bgColB-=0.001;

if(ballX<-0.3)

{

flag=1;

}

}

glutPostRedisplay(); //Tell GLUT that the display has changed

//Tell GLUT to call update again in 25 milliseconds

glutTimerFunc(25, update, 0);

}

int main(int argc,char\*\* argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB|GLUT\_DEPTH);

glutInitWindowSize(400,400);

glutCreateWindow("Sun");

initRendering();

glutDisplayFunc(drawScene);

glutFullScreen();

glutSpecialFunc(keyPress);

glutReshapeFunc(handleResize);

glutTimerFunc(25, update, 0);

glutMainLoop();

return(0);

}

Output :-

1. For Sunrise:-

}

int main(int argc,char\*\* argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB|GLUT\_DEPTH);

glutInitWindowSize(400,400);

glutCreateWindow("Sun");

initRendering();

glutDisplayFunc(drawScene);

glutFullScreen();

glutSpecialFunc(keyPress);

glutReshapeFunc(handleResize);

glutTimerFunc(25, update, 0);

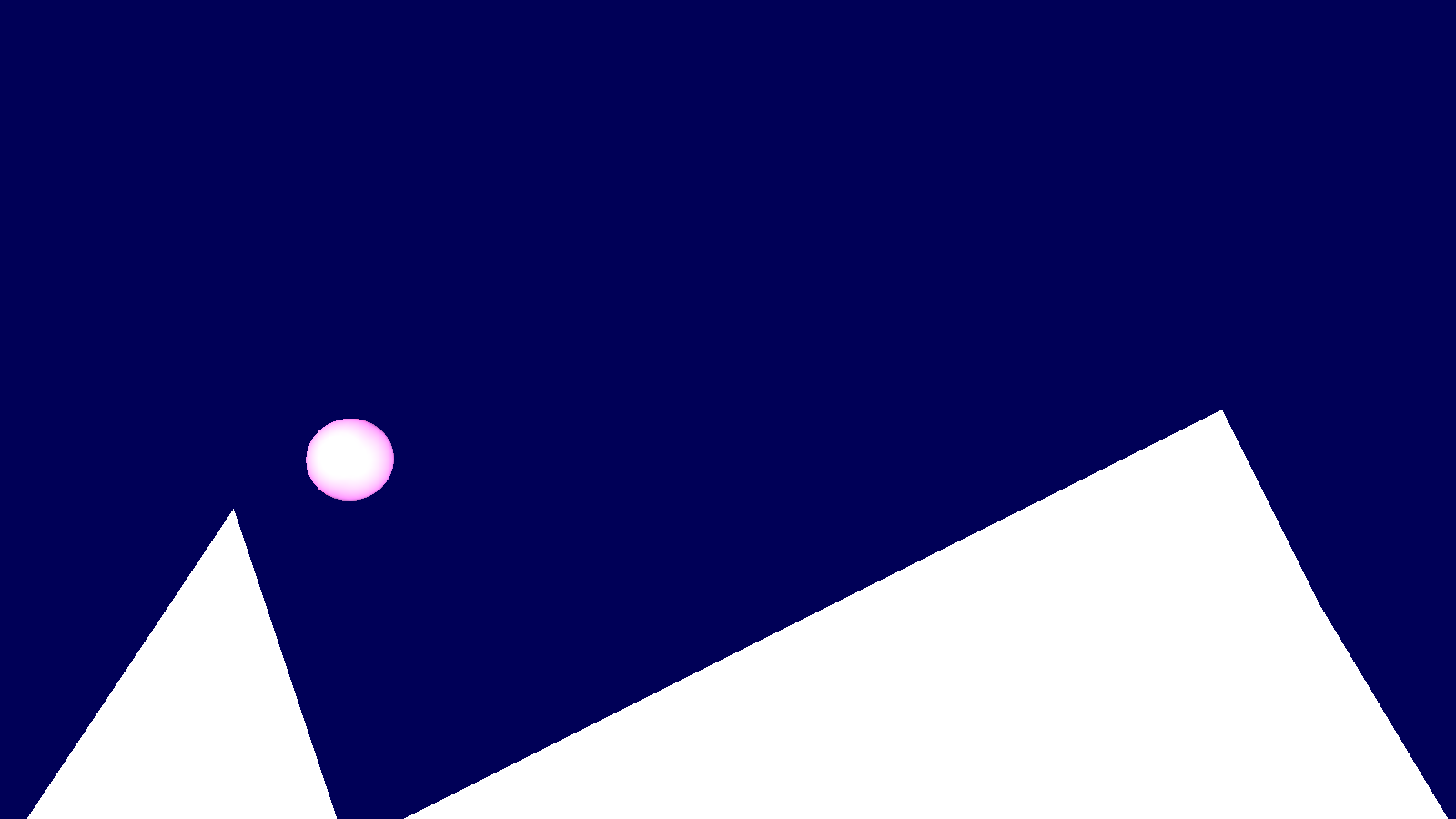
glutMainLoop();

return(0);

}

Output :-

1. For Sunrise:-
2. For Sunset:-



**Name** – Tejas Pravin Pawar ( SEB25 )

**Assignment no – 3**

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

**Source Code :-**

#include <iostream>

# include <graphics.h>

# include <stdlib.h>

using namespace std;

class dcircle

{

private: int x0, y0;

public:

dcircle()

{

x0=0;

y0=0;

}

void setoff(int xx, int yy)

{

x0=xx;

y0=yy;

}

void drawc(int x1, int y1, int r)

{

float d;

int x,y;

x=0;

y=r;

d=3-2\*r;

do

{

putpixel(x1+x0+x, y0+y-y1, 15);

putpixel(x1+x0+y, y0+x-y1,15);

putpixel(x1+x0+y, y0-x-y1,15);

putpixel(x1+x0+x,y0-y-y1,15);

putpixel(x1+x0-x,y0-y-y1,15);

putpixel(x1+x0-y, y0-x-y1,15);

putpixel(x1+x0-y, y0+x-y1,15);

putpixel(x1+x0-x, y0+y-y1,15);

if (d<=0)

{

d = d+4\*x+6;

}

else

{

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

y=y-1;

}

x=x+1;

}

while(x<y);

}

};

class pt

{

protected: int xco, yco,color;

public:

pt()

{

xco=0,yco=0,color=15;

}

void setco(int x, int y)

{

xco=x;

yco=y;

}

void setcolor(int c)

{

color=c;

}

void draw()

{

putpixel(xco,yco,color);

}

};

class dline:public pt

{

private: int x2, y2;

public:

dline():pt()

{

x2=0;

y2=0;

}

void setline(int x, int y, int xx, int yy)

{

pt::setco(x,y);

x2=xx;

y2=yy;

}

void drawl( int colour)

{

float x,y,dx,dy,length;

int i;

pt::setcolor(colour);

dx= abs(x2-xco);

dy=abs(y2-yco);

if(dx>=dy)

{

length= dx;

}

else

{

length= dy;

}

dx=(x2-xco)/length;

dy=(y2-yco)/length;

x=xco+0.5;

y=yco+0.5;

i=1;

while(i<=length)

{

pt::setco(x,y);

pt::draw();

x=x+dx;

y=y+dy;

i=i+1;

}

pt::setco(x,y);

pt::draw();

}

};

int main()

{

int gd=DETECT, gm;

initgraph(&gd, &gm, NULL);

int x,y,r, x1, x2, y1, y2, xmax, ymax, xmid, ymid, n, i;

dcircle c;

cout<<"\nenter coordinates of centre of circle : ";

cout<<"\n enter the value of x : ";

cin>>x;

cout<<"\nenter the value of y : ";

cin>>y;

cout<<"\nenter the value of radius : ";

cin>>r;

xmax= getmaxx();

ymax=getmaxy();

xmid=xmax/2;

ymid=ymax/2;

setcolor(1);

c.setoff(xmid,ymid);

line(xmid, 0, xmid, ymax);

line(0,ymid,xmax,ymid);

setcolor(15);

c.drawc(x,y,r);

pt p1;

p1.setco(100,100);

p1.setcolor(14);

dline l;

l.setline(x1+xmid, ymid-y1, x2+xmid, ymid-y2);

cout<<"Enter Total Number of lines : ";

cin>>n;

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

{

cout<<"Enter co-ordinates of point x1 : ";

cin>>x1;

cout<<"enter coordinates of point y1 : ";

cin>>y1;

cout<<"Enter co-ordinates of point x2 : ";

cin>>x2;

cout<<"enter coordinates of point y2 : ";

cin>>y2;

l.setline(x1+xmid, ymid-y1, x2+xmid, ymid-y2);

l.drawl(15);

}

cout<<"\nEnter coordinates of centre of circle : ";

cout<<"\n Enter the value of x : ";

cin>>x;

cout<<"\nEnter the value of y : ";

cin>>y;

cout<<"\nEnter the value of radius : ";

cin>>r;

setcolor(5);

c.drawc(x,y,r);

getch();

delay(200);

closegraph();

return 0;

}

Input :

For circle 1 (internal circle):

Value Of X : 100

Value Of Y : 70

Value Of R : 30

Total Number of lines: 3

Line No.1:

Value of X1: 40

Value of Y1: 40

Value of X2: 100

Value of Y2: 124

Line No.2:

Value of X1: 40

Value of Y1: 40

Value of X2: 160

Value of Y2: 40

Line No.3:

Value of X1: 40

Value of Y1: 40

Value of X2: 100

Value of Y2: 124

For Circle No.2 (External circle):

Value of X: 100

Value of Y: 62

Value of R: 60

Output 1:

putpixel(x1+x0-y, y0-x-y1,15);

putpixel(x1+x0-y, y0+x-y1,15);

putpixel(x1+x0-x, y0+y-y1,15);

if (d<=0)

{

d = d+4\*x+6;

}

else

{

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

y=y-1;

}

x=x+1;

}

while(x<y);

}

};

class pt

{

protected: int xco, yco,color;

public:

pt()

{

xco=0,yco=0,color=15;

}

void setco(int x, int y)

{

xco=x;

yco=y;

}

void setcolor(int c)

{

color=c;

}

void draw()

{

putpixel(xco,yco,color);

}

};

class dline:public pt

{

private: int x2, y2;

public:

dline():pt()

{

x2=0;

y2=0;

}

void setline(int x, int y, int xx, int yy)

{

pt::setco(x,y);

x2=xx;

y2=yy;

}

void drawl( int colour)

{

float x,y,dx,dy,length;

int i;

pt::setcolor(colour);

dx= abs(x2-xco);

dy=abs(y2-yco);

if(dx>=dy)

{

length= dx;

}

else

{

length= dy;

}

dx=(x2-xco)/length;

dy=(y2-yco)/length;

x=xco+0.5;

y=yco+0.5;

i=1;

while(i<=length)

{

pt::setco(x,y);

pt::draw();

x=x+dx;

y=y+dy;

i=i+1;

}

pt::setco(x,y);

pt::draw();

}

};

int main()

{

int gd=DETECT, gm;

initgraph(&gd, &gm, NULL);

int x,y,r, x1, x2, y1, y2, xmax, ymax, xmid, ymid, n, i;

dcircle c;

cout<<"\nenter coordinates of centre of circle : ";

cout<<"\n enter the value of x : ";

cin>>x;

cout<<"\nenter the value of y : ";

cin>>y;

cout<<"\nenter the value of radius : ";

cin>>r;

xmax= getmaxx();

ymax=getmaxy();

xmid=xmax/2;

ymid=ymax/2;

setcolor(1);

c.setoff(xmid,ymid);

line(xmid, 0, xmid, ymax);

line(0,ymid,xmax,ymid);

setcolor(15);

c.drawc(x,y,r);

pt p1;

p1.setco(100,100);

p1.setcolor(14);

dline l;

l.setline(x1+xmid, ymid-y1, x2+xmid, ymid-y2);

cout<<"Enter Total Number of lines : ";

cin>>n;

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

{

cout<<"Enter co-ordinates of point x1 : ";

cin>>x1;

cout<<"enter coordinates of point y1 : ";

cin>>y1;

cout<<"Enter co-ordinates of point x2 : ";

cin>>x2;

cout<<"enter coordinates of point y2 : ";

cin>>y2;

l.setline(x1+xmid, ymid-y1, x2+xmid, ymid-y2);

l.drawl(15);

}

cout<<"\nEnter coordinates of centre of circle : ";

cout<<"\n Enter the value of x : ";

cin>>x;

cout<<"\nEnter the value of y : ";

cin>>y;

cout<<"\nEnter the value of radius : ";

cin>>r;

setcolor(5);

c.drawc(x,y,r);

getch();

delay(200);

closegraph();

return 0;

}

Input :

For circle 1 (internal circle):

Value Of X : 100

Value Of Y : 70

Value Of R : 30

Total Number of lines: 3

Line No.1:

Value of X1: 40

Value of Y1: 40

Value of X2: 100

Value of Y2: 124

Line No.2:

Value of X1: 40

Value of Y1: 40

Value of X2: 160

Value of Y2: 40

Line No.3:

Value of X1: 40

Value of Y1: 40

Value of X2: 100

Value of Y2: 124

For Circle No.2 (External circle):

Value of X: 100

Value of Y: 62

Value of R: 60

Output 1:

{

private: int x2, y2;

public:

dline():pt()

{

x2=0;

y2=0;

}

void setline(int x, int y, int xx, int yy)

{

pt::setco(x,y);

x2=xx;

y2=yy;

}

void drawl( int colour)

{

float x,y,dx,dy,length;

int i;

pt::setcolor(colour);

dx= abs(x2-xco);

dy=abs(y2-yco);

if(dx>=dy)

{

length= dx;

}

else

{

length= dy;

}

dx=(x2-xco)/length;

dy=(y2-yco)/length;

x=xco+0.5;

y=yco+0.5;

i=1;

while(i<=length)

{

pt::setco(x,y);

pt::draw();

x=x+dx;

y=y+dy;

i=i+1;

}

pt::setco(x,y);

pt::draw();

}

};

int main()

{

int gd=DETECT, gm;

initgraph(&gd, &gm, NULL);

int x,y,r, x1, x2, y1, y2, xmax, ymax, xmid, ymid, n, i;

dcircle c;

cout<<"\nenter coordinates of centre of circle : ";

cout<<"\n enter the value of x : ";

cin>>x;

cout<<"\nenter the value of y : ";

cin>>y;

cout<<"\nenter the value of radius : ";

cin>>r;

xmax= getmaxx();

ymax=getmaxy();

xmid=xmax/2;

ymid=ymax/2;

setcolor(1);

c.setoff(xmid,ymid);

line(xmid, 0, xmid, ymax);

line(0,ymid,xmax,ymid);

setcolor(15);

c.drawc(x,y,r);

pt p1;

p1.setco(100,100);

p1.setcolor(14);

dline l;

l.setline(x1+xmid, ymid-y1, x2+xmid, ymid-y2);

cout<<"Enter Total Number of lines : ";

cin>>n;

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

{

cout<<"Enter co-ordinates of point x1 : ";

cin>>x1;

cout<<"enter coordinates of point y1 : ";

cin>>y1;

cout<<"Enter co-ordinates of point x2 : ";

cin>>x2;

cout<<"enter coordinates of point y2 : ";

cin>>y2;

l.setline(x1+xmid, ymid-y1, x2+xmid, ymid-y2);

l.drawl(15);

}

cout<<"\nEnter coordinates of centre of circle : ";

cout<<"\n Enter the value of x : ";

cin>>x;

cout<<"\nEnter the value of y : ";

cin>>y;

cout<<"\nEnter the value of radius : ";

cin>>r;

setcolor(5);

c.drawc(x,y,r);

getch();

delay(200);

closegraph();

return 0;

}

Input :

For circle 1 (internal circle):

Value Of X : 100

Value Of Y : 70

Value Of R : 30

Total Number of lines: 3

Line No.1:

Value of X1: 40

Value of Y1: 40

Value of X2: 100

Value of Y2: 124

Line No.2:

Value of X1: 40

Value of Y1: 40

Value of X2: 160

Value of Y2: 40

Line No.3:

Value of X1: 40

Value of Y1: 40

Value of X2: 100

Value of Y2: 124

For Circle No.2 (External circle):

Value of X: 100

Value of Y: 62

Value of R: 60

Output 1:

i=i+1;

}

pt::setco(x,y);

pt::draw();

}

};

int main()

{

int gd=DETECT, gm;

initgraph(&gd, &gm, NULL);

int x,y,r, x1, x2, y1, y2, xmax, ymax, xmid, ymid, n, i;

dcircle c;

cout<<"\nenter coordinates of centre of circle : ";

cout<<"\n enter the value of x : ";

cin>>x;

cout<<"\nenter the value of y : ";

cin>>y;

cout<<"\nenter the value of radius : ";

cin>>r;

xmax= getmaxx();

ymax=getmaxy();

xmid=xmax/2;

ymid=ymax/2;

setcolor(1);

c.setoff(xmid,ymid);

line(xmid, 0, xmid, ymax);

line(0,ymid,xmax,ymid);

setcolor(15);

c.drawc(x,y,r);

pt p1;

p1.setco(100,100);

p1.setcolor(14);

dline l;

l.setline(x1+xmid, ymid-y1, x2+xmid, ymid-y2);

cout<<"Enter Total Number of lines : ";

cin>>n;

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

{

cout<<"Enter co-ordinates of point x1 : ";

cin>>x1;

cout<<"enter coordinates of point y1 : ";

cin>>y1;

cout<<"Enter co-ordinates of point x2 : ";

cin>>x2;

cout<<"enter coordinates of point y2 : ";

cin>>y2;

l.setline(x1+xmid, ymid-y1, x2+xmid, ymid-y2);

l.drawl(15);

}

cout<<"\nEnter coordinates of centre of circle : ";

cout<<"\n Enter the value of x : ";

cin>>x;

cout<<"\nEnter the value of y : ";

cin>>y;

cout<<"\nEnter the value of radius : ";

cin>>r;

setcolor(5);

c.drawc(x,y,r);

getch();

delay(200);

closegraph();

return 0;

}

Input :

For circle 1 (internal circle):

Value Of X : 100

Value Of Y : 70

Value Of R : 30

Total Number of lines: 3

Line No.1:

Value of X1: 40

Value of Y1: 40

Value of X2: 100

Value of Y2: 124

Line No.2:

Value of X1: 40

Value of Y1: 40

Value of X2: 160

Value of Y2: 40

Line No.3:

Value of X1: 40

Value of Y1: 40

Value of X2: 100

Value of Y2: 124

For Circle No.2 (External circle):

Value of X: 100

Value of Y: 62

Value of R: 60

Output 1:

cin>>x1;

cout<<"enter coordinates of point y1 : ";

cin>>y1;

cout<<"Enter co-ordinates of point x2 : ";

cin>>x2;

cout<<"enter coordinates of point y2 : ";

cin>>y2;

l.setline(x1+xmid, ymid-y1, x2+xmid, ymid-y2);

l.drawl(15);

}

cout<<"\nEnter coordinates of centre of circle : ";

cout<<"\n Enter the value of x : ";

cin>>x;

cout<<"\nEnter the value of y : ";

cin>>y;

cout<<"\nEnter the value of radius : ";

cin>>r;

setcolor(5);

c.drawc(x,y,r);

getch();

delay(200);

closegraph();

return 0;

}

**Input** :

For circle 1 (internal circle):

Value Of X : 100

Value Of Y : 70

Value Of R : 30

Total Number of lines: 3

Line No.1:

Value of X1: 40

Value of Y1: 40

Value of X2: 100

Value of Y2: 124

Line No.2:

Value of X1: 40

Value of Y1: 40

Value of X2: 160

Value of Y2: 40

Line No.3:

Value of X1: 40

Value of Y1: 40

Value of X2: 100

Value of Y2: 124

For Circle No.2 (External circle):

Value of X: 100

Value of Y: 62

Value of R: 60

Output 1:

Value of Y2: 40

Line No.3:

Value of X1: 40

Value of Y1: 40

Value of X2: 100

Value of Y2: 124

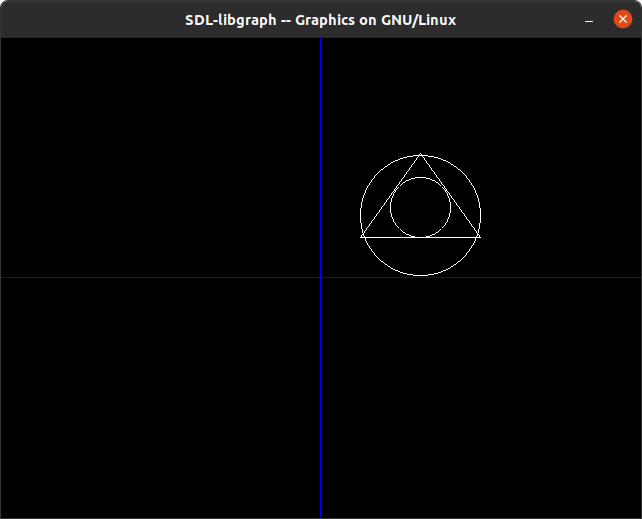
For Circle No.2 (External circle):

Value of X: 100

Value of Y: 62

Value of R: 60

**Output** :



**Name** – Tejas Pravin Pawar ( SEB25 )

**Assignment no – 2**

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

**Source code :**

#include<iostream>

#include<stdlib.h>

#include<math.h>

#include<graphics.h>

#include<dos.h>

using namespace std;

class Coordinate

{

public:

int x,y;

char code[4];

};

class Lineclip

{

public:

Coordinate PT;

void drawwindow();

void drawline(Coordinate p1,Coordinate p2);

Coordinate setcode(Coordinate p);

int visibility(Coordinate p1,Coordinate p2);

Coordinate resetendpt(Coordinate p1,Coordinate p2);

};

int main()

{

Lineclip lc;

int gd = DETECT,v,gm;

Coordinate p1,p2,p3,p4,ptemp;

cout<<"\n Enter x1 and y1\n";

cin>>p1.x>>p1.y;

cout<<"\n Enter x2 and y2\n";

cin>>p2.x>>p2.y;

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

lc.drawwindow();

delay(2000);

lc.drawline (p1,p2);

delay(2000);

cleardevice();

delay(2000);

p1=lc.setcode(p1);

p2=lc.setcode(p2);

v=lc.visibility(p1,p2);

delay(2000);

switch(v)

{

case 0: lc.drawwindow();

delay(2000);

lc.drawline(p1,p2);

break;

case 1:lc.drawwindow();

delay(2000);

break;

case 2:p3=lc.resetendpt(p1,p2);

p4=lc.resetendpt(p2,p1);

lc.drawwindow();

delay(2000);

lc.drawline(p3,p4);

break;

}

delay(2000);

closegraph();

}

void Lineclip::drawwindow()

{

line(150,100,450,100);

line(450,100,450,350);

line(450,350,150,350);

line(150,350,150,100);

}

void Lineclip::drawline(Coordinate p1,Coordinate p2)

{

line(p1.x,p1.y,p2.x,p2.y);

}

Coordinate Lineclip::setcode(Coordinate p)

{

Coordinate ptemp;

if(p.y<100)

ptemp.code[0]='1';

else

ptemp.code[0]='0';

if(p.y>350)

ptemp.code[1]='1';

else

ptemp.code[1]='0';

if(p.x>450)

ptemp.code[2]='1';

else

ptemp.code[2]='0';

if(p.x<150)

ptemp.code[3]='1';

else

ptemp.code[3]='0';

ptemp.x=p.x;

ptemp.y=p.y;

return(ptemp);

};

int Lineclip:: visibility(Coordinate p1,Coordinate p2)

{

int i,flag=0;

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

{

if(p1.code[i]!='0' || (p2.code[i]=='1'))

flag='0';

}

if(flag==0)

return(0);

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

{

if(p1.code[i]==p2.code[i] && (p2.code[i]=='1'))

flag='0';

}

if(flag==0)

return(1);

return(2);

}

Coordinate Lineclip::resetendpt(Coordinate p1,Coordinate p2)

{

Coordinate temp;

int x,y,i;

float m,k;

if(p1.code[3]=='1')

x=150;

if(p1.code[2]=='1')

x=450;

if((p1.code[3]=='1') || (p1.code[2])=='1')

{

m=(float)(p2.y-p1.y)/(p2.x-p1.x);

k=(p1.y+(m\*(x-p1.x)));

temp.y=k;

temp.x=x;

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

temp.code[i]=p1.code[i];

if(temp.y<=350 && temp.y>=100)

return (temp);

}

if(p1.code[0]=='1')

y=100;

if(p1.code[1]=='1')

y=350;

if((p1.code[1]=='1') || (p1.code[1]=='1'))

{

m=(float)(p2.y-p1.y)/(p2.x-p1.x);

k=(float)p1.x+(float)(y-p1.y)/m;

temp.x=k;

temp.y=y;

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

temp.code[i]=p1.code[i];

return(temp);

}

else

return(p1);

}

Input :

x1 , y1:

200

200

x2, y2:

600

100

Output :

lc.drawline (p1,p2);

delay(2000);

cleardevice();

delay(2000);

p1=lc.setcode(p1);

p2=lc.setcode(p2);

v=lc.visibility(p1,p2);

delay(2000);

switch(v)

{

case 0: lc.drawwindow();

delay(2000);

lc.drawline(p1,p2);

break;

case 1:lc.drawwindow();

delay(2000);

break;

case 2:p3=lc.resetendpt(p1,p2);

p4=lc.resetendpt(p2,p1);

lc.drawwindow();

delay(2000);

lc.drawline(p3,p4);

break;

}

delay(2000);

closegraph();

}

void Lineclip::drawwindow()

{

line(150,100,450,100);

line(450,100,450,350);

line(450,350,150,350);

line(150,350,150,100);

}

void Lineclip::drawline(Coordinate p1,Coordinate p2)

{

line(p1.x,p1.y,p2.x,p2.y);

}

Coordinate Lineclip::setcode(Coordinate p)

{

Coordinate ptemp;

if(p.y<100)

ptemp.code[0]='1';

else

ptemp.code[0]='0';

if(p.y>350)

ptemp.code[1]='1';

else

ptemp.code[1]='0';

if(p.x>450)

ptemp.code[2]='1';

else

ptemp.code[2]='0';

if(p.x<150)

ptemp.code[3]='1';

else

ptemp.code[3]='0';

ptemp.x=p.x;

ptemp.y=p.y;

return(ptemp);

};

int Lineclip:: visibility(Coordinate p1,Coordinate p2)

{

int i,flag=0;

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

{

if(p1.code[i]!='0' || (p2.code[i]=='1'))

flag='0';

}

if(flag==0)

return(0);

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

{

if(p1.code[i]==p2.code[i] && (p2.code[i]=='1'))

flag='0';

}

if(flag==0)

return(1);

return(2);

}

Coordinate Lineclip::resetendpt(Coordinate p1,Coordinate p2)

{

Coordinate temp;

int x,y,i;

float m,k;

if(p1.code[3]=='1')

x=150;

if(p1.code[2]=='1')

x=450;

if((p1.code[3]=='1') || (p1.code[2])=='1')

{

m=(float)(p2.y-p1.y)/(p2.x-p1.x);

k=(p1.y+(m\*(x-p1.x)));

temp.y=k;

temp.x=x;

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

temp.code[i]=p1.code[i];

if(temp.y<=350 && temp.y>=100)

return (temp);

}

if(p1.code[0]=='1')

y=100;

if(p1.code[1]=='1')

y=350;

if((p1.code[1]=='1') || (p1.code[1]=='1'))

{

m=(float)(p2.y-p1.y)/(p2.x-p1.x);

k=(float)p1.x+(float)(y-p1.y)/m;

temp.x=k;

temp.y=y;

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

temp.code[i]=p1.code[i];

return(temp);

}

else

return(p1);

}

Input :

x1 , y1:

200

200

x2, y2:

600

100

Output :

{

Coordinate ptemp;

if(p.y<100)

ptemp.code[0]='1';

else

ptemp.code[0]='0';

if(p.y>350)

ptemp.code[1]='1';

else

ptemp.code[1]='0';

if(p.x>450)

ptemp.code[2]='1';

else

ptemp.code[2]='0';

if(p.x<150)

ptemp.code[3]='1';

else

ptemp.code[3]='0';

ptemp.x=p.x;

ptemp.y=p.y;

return(ptemp);

};

int Lineclip:: visibility(Coordinate p1,Coordinate p2)

{

int i,flag=0;

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

{

if(p1.code[i]!='0' || (p2.code[i]=='1'))

flag='0';

}

if(flag==0)

return(0);

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

{

if(p1.code[i]==p2.code[i] && (p2.code[i]=='1'))

flag='0';

}

if(flag==0)

return(1);

return(2);

}

Coordinate Lineclip::resetendpt(Coordinate p1,Coordinate p2)

{

Coordinate temp;

int x,y,i;

float m,k;

if(p1.code[3]=='1')

x=150;

if(p1.code[2]=='1')

x=450;

if((p1.code[3]=='1') || (p1.code[2])=='1')

{

m=(float)(p2.y-p1.y)/(p2.x-p1.x);

k=(p1.y+(m\*(x-p1.x)));

temp.y=k;

temp.x=x;

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

temp.code[i]=p1.code[i];

if(temp.y<=350 && temp.y>=100)

return (temp);

}

if(p1.code[0]=='1')

y=100;

if(p1.code[1]=='1')

y=350;

if((p1.code[1]=='1') || (p1.code[1]=='1'))

{

m=(float)(p2.y-p1.y)/(p2.x-p1.x);

k=(float)p1.x+(float)(y-p1.y)/m;

temp.x=k;

temp.y=y;

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

temp.code[i]=p1.code[i];

return(temp);

}

else

return(p1);

}

Input :

x1 , y1:

200

200

x2, y2:

600

100

Output :

return(2);

}

Coordinate Lineclip::resetendpt(Coordinate p1,Coordinate p2)

{

Coordinate temp;

int x,y,i;

float m,k;

if(p1.code[3]=='1')

x=150;

if(p1.code[2]=='1')

x=450;

if((p1.code[3]=='1') || (p1.code[2])=='1')

{

m=(float)(p2.y-p1.y)/(p2.x-p1.x);

k=(p1.y+(m\*(x-p1.x)));

temp.y=k;

temp.x=x;

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

temp.code[i]=p1.code[i];

if(temp.y<=350 && temp.y>=100)

return (temp);

}

if(p1.code[0]=='1')

y=100;

if(p1.code[1]=='1')

y=350;

if((p1.code[1]=='1') || (p1.code[1]=='1'))

{

m=(float)(p2.y-p1.y)/(p2.x-p1.x);

k=(float)p1.x+(float)(y-p1.y)/m;

temp.x=k;

temp.y=y;

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

temp.code[i]=p1.code[i];

return(temp);

}

else

return(p1);

}

Input :

x1 , y1:

200

200

x2, y2:

600

100

Output :

x1 , y1:

200

200

x2, y2:

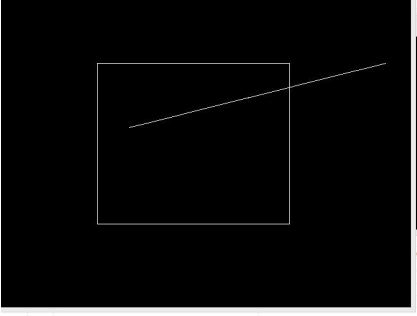
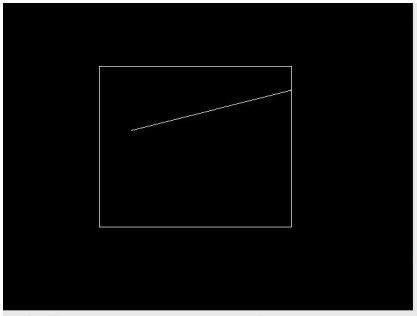
600

100

**Output** :

Whithout clipping :

With clipping :



**Name** – Tejas Pravin Pawar ( SEB25 )

**Assignment no – 1**

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

**Source code :**

#include <conio.h>

#include <iostream>

#include <graphics.h>

#include <stdlib.h>

using namespace std;

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 Scan Fill Algorithm ";

cout<<"\n Enter Number Of Vertices Of Polygon: ";

cin>>v;

if(v>2)

{

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

{

cout<<"\nEnter 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.";

}

void poly::calcs()

{

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;

}

}

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)

{

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

{

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

x=x+x1;

}

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

inter[c++]=x;

}

}

}

void poly::sort(int z) // sorting

{

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);

}

delay(100);

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

{

delay(100);

line(inter[i],z,inter[i+1],z);

}

}

int main() //main

{

int cl;

initwindow(500,600);

cleardevice();

poly x;

x.read();

x.calcs();

cleardevice();

cout<<"\n\tEnter The Color You Want :(In Range 0 To 15 )->"; //selecting color

cin>>cl;

setcolor(cl);

x.display();

closegraph(); //closing graph

getch();

return 0;

}

Input:-

Number of Vertices : 4

Coordinate 1st :

X1 = 200

Y1 = 200

Coordinate 2nd :

X2 = 200

Y2 = 400

Coordinate 3rd

:

X3 = 400

Y3 = 200

Coordinate 4th

:

X4 = 400

Y4 = 400

Output:

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

{

cout<<"\nEnter 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.";

}

void poly::calcs()

{

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;

}

}

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)

{

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

{

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

x=x+x1;

}

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

inter[c++]=x;

}

}

}

void poly::sort(int z) // sorting

{

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);

}

delay(100);

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

{

delay(100);

line(inter[i],z,inter[i+1],z);

}

}

int main() //main

{

int cl;

initwindow(500,600);

cleardevice();

poly x;

x.read();

x.calcs();

cleardevice();

cout<<"\n\tEnter The Color You Want :(In Range 0 To 15 )->"; //selecting color

cin>>cl;

setcolor(cl);

x.display();

closegraph(); //closing graph

getch();

return 0;

}

Input:-

Number of Vertices : 4

Coordinate 1st :

X1 = 200

Y1 = 200

Coordinate 2nd :

X2 = 200

Y2 = 400

Coordinate 3rd

:

X3 = 400

Y3 = 200

Coordinate 4th

:

X4 = 400

Y4 = 400

Output:

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)

{

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

{

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

x=x+x1;

}

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

inter[c++]=x;

}

}

}

void poly::sort(int z) // sorting

{

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);

}

delay(100);

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

{

delay(100);

line(inter[i],z,inter[i+1],z);

}

}

int main() //main

{

int cl;

initwindow(500,600);

cleardevice();

poly x;

x.read();

x.calcs();

cleardevice();

cout<<"\n\tEnter The Color You Want :(In Range 0 To 15 )->"; //selecting color

cin>>cl;

setcolor(cl);

x.display();

closegraph(); //closing graph

getch();

return 0;

}

Input:-

Number of Vertices : 4

Coordinate 1st :

X1 = 200

Y1 = 200

Coordinate 2nd :

X2 = 200

Y2 = 400

Coordinate 3rd

:

X3 = 400

Y3 = 200

Coordinate 4th

:

X4 = 400

Y4 = 400

Output:

y1=y2;

y2=temp;

}

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

{

if((y1-y2)==0)

x=x1;

else

{

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

x=x+x1;

}

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

inter[c++]=x;

}

}

}

void poly::sort(int z) // sorting

{

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);

}

delay(100);

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

{

delay(100);

line(inter[i],z,inter[i+1],z);

}

}

int main() //main

{

int cl;

initwindow(500,600);

cleardevice();

poly x;

x.read();

x.calcs();

cleardevice();

cout<<"\n\tEnter The Color You Want :(In Range 0 To 15 )->"; //selecting color

cin>>cl;

setcolor(cl);

x.display();

closegraph(); //closing graph

getch();

return 0;

}

Input:-

Number of Vertices : 4

Coordinate 1st :

X1 = 200

Y1 = 200

Coordinate 2nd :

X2 = 200

Y2 = 400

Coordinate 3rd

:

X3 = 400

Y3 = 200

Coordinate 4th

:

X4 = 400

Y4 = 400

Output:

cout<<"\n\tEnter The Color You Want :(In Range 0 To 15 )->"; //selecting color

cin>>cl;

setcolor(cl);

x.display();

closegraph(); //closing graph

getch();

return 0;

}

Input:-

Number of Vertices : 4

Coordinate 1st :

X1 = 200

Y1 = 200

Coordinate 2nd :

X2 = 200

Y2 = 400

Coordinate 3rd

:

X3 = 400

Y3 = 200

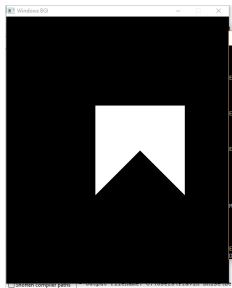
Coordinate 4th

:

X4 = 400

Y4 = 400

**Output**:



**Name** – Tejas Pravin Pawar ( SEB25 )

**Assignment no – 4**

**Title :** Write C++ program to draw 2-D object and perform following basic transformation

**Source code :**

#include<iostream>

#include<graphics.h>

#include<math.h>

using namespace std;

class transform

{

public:

int m,a[20][20],c[20][20];

int i,j,k;

public:

void object();

void accept();

void operator \*(float b[20][20])

{

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

{

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

{

c[i][j]=0;

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

{

c[i][j]=c[i][j]+(a[i][k]\*b[k][j]);

}}}}};

void transform::object()

{

int gd,gm;

gd=DETECT;

initgraph(&gd,&gm,NULL);

line(300,0,300,600);

line(0,300,600,300);

for( i=0;i<m-1;i++)

{

line(300+a[i][0],300-a[i][1],300+a[i+1][0],300-a[i+1][1]);

}

line(300+a[0][0],300-a[0][1],300+a[i][0],300-a[i][1]);

for( i=0;i<m-1;i++)

{

line(300+c[i][0],300-c[i][1],300+c[i+1][0],300-c[i+1][1]);

}

line(300+c[0][0],300-c[0][1],300+c[i][0],300-c[i][1]);

int temp;

cout << "Press 1 to continue";

cin >> temp;

closegraph();

}

void transform::accept()

{

cout<<"\n";

cout<<"Enter the Number Of Edges:";

cin>>m;

cout<<"\nEnter The Coordinates :";

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

{

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

{

if(j>=2)

a[i][j]=1;

else

cin>>a[i][j];

}

}

}

int main()

{

int ch,tx,ty,sx,sy;

float deg,theta,b[20][20];

transform t;

t.accept();

cout<<"\nEnter your choice";

cout<<"\n1.Translation"

"\n2.Scaling"

"\n3.Rotation";

cin>>ch;

switch(ch)

{

case 1: cout<<"\nTRANSLATION OPERATION\n";

cout<<"Enter value for tx and ty:";

cin>>tx>>ty;

b[0][0]=b[2][2]=b[1][1]=1;

b[0][1]=b[0][2]=b[1][0]=b[1][2]=0;

b[2][0]=tx;

b[2][1]=ty;

t \* b;

t.object();

break;

case 2: cout<<"\nSCALING OPERATION\n";

cout<<"Enter value for sx,sy:";

cin>>sx>>sy;

b[0][0]=sx;

b[1][1]=sy;

b[0][1]=b[0][2]=b[1][0]=b[1][2]=0;

b[2][0]=b[2][1]=0;

b[2][2] = 1;

t \* b;

t.object();

break;

case 3: cout<<"\nROTATION OPERATION\n";

cout<<"Enter value for angle:";

cin>>deg;

theta=deg\*(3.14/100);

b[0][0]=b[1][1]=cos(theta);

b[0][1]=sin(theta);

b[1][0]=sin(-theta);

b[0][2]=b[1][2]=b[2][0]=b[2][1]=0;

b[2][2]=1;

t \* b;

t.object();

break;

default:

cout<<"\nInvalid choice";

}

getch();

return 0;

}

Input:-

Provided in image given below.

Output:

For Translation:

line(300+a[i][0],300-a[i][1],300+a[i+1][0],300-a[i+1][1]);

}

line(300+a[0][0],300-a[0][1],300+a[i][0],300-a[i][1]);

for( i=0;i<m-1;i++)

{

line(300+c[i][0],300-c[i][1],300+c[i+1][0],300-c[i+1][1]);

}

line(300+c[0][0],300-c[0][1],300+c[i][0],300-c[i][1]);

int temp;

cout << "Press 1 to continue";

cin >> temp;

closegraph();

}

void transform::accept()

{

cout<<"\n";

cout<<"Enter the Number Of Edges:";

cin>>m;

cout<<"\nEnter The Coordinates :";

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

{

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

{

if(j>=2)

a[i][j]=1;

else

cin>>a[i][j];

}

}

}

int main()

{

int ch,tx,ty,sx,sy;

float deg,theta,b[20][20];

transform t;

t.accept();

cout<<"\nEnter your choice";

cout<<"\n1.Translation"

"\n2.Scaling"

"\n3.Rotation";

cin>>ch;

switch(ch)

{

case 1: cout<<"\nTRANSLATION OPERATION\n";

cout<<"Enter value for tx and ty:";

cin>>tx>>ty;

b[0][0]=b[2][2]=b[1][1]=1;

b[0][1]=b[0][2]=b[1][0]=b[1][2]=0;

b[2][0]=tx;

b[2][1]=ty;

t \* b;

t.object();

break;

case 2: cout<<"\nSCALING OPERATION\n";

cout<<"Enter value for sx,sy:";

cin>>sx>>sy;

b[0][0]=sx;

b[1][1]=sy;

b[0][1]=b[0][2]=b[1][0]=b[1][2]=0;

b[2][0]=b[2][1]=0;

b[2][2] = 1;

t \* b;

t.object();

break;

case 3: cout<<"\nROTATION OPERATION\n";

cout<<"Enter value for angle:";

cin>>deg;

theta=deg\*(3.14/100);

b[0][0]=b[1][1]=cos(theta);

b[0][1]=sin(theta);

b[1][0]=sin(-theta);

b[0][2]=b[1][2]=b[2][0]=b[2][1]=0;

b[2][2]=1;

t \* b;

t.object();

break;

default:

cout<<"\nInvalid choice";

}

getch();

return 0;

}

Input:-

Provided in image given below.

Output:

For Translation:

cout<<"\nEnter your choice";

cout<<"\n1.Translation"

"\n2.Scaling"

"\n3.Rotation";

cin>>ch;

switch(ch)

{

case 1: cout<<"\nTRANSLATION OPERATION\n";

cout<<"Enter value for tx and ty:";

cin>>tx>>ty;

b[0][0]=b[2][2]=b[1][1]=1;

b[0][1]=b[0][2]=b[1][0]=b[1][2]=0;

b[2][0]=tx;

b[2][1]=ty;

t \* b;

t.object();

break;

case 2: cout<<"\nSCALING OPERATION\n";

cout<<"Enter value for sx,sy:";

cin>>sx>>sy;

b[0][0]=sx;

b[1][1]=sy;

b[0][1]=b[0][2]=b[1][0]=b[1][2]=0;

b[2][0]=b[2][1]=0;

b[2][2] = 1;

t \* b;

t.object();

break;

case 3: cout<<"\nROTATION OPERATION\n";

cout<<"Enter value for angle:";

cin>>deg;

theta=deg\*(3.14/100);

b[0][0]=b[1][1]=cos(theta);

b[0][1]=sin(theta);

b[1][0]=sin(-theta);

b[0][2]=b[1][2]=b[2][0]=b[2][1]=0;

b[2][2]=1;

t \* b;

t.object();

break;

default:

cout<<"\nInvalid choice";

}

getch();

return 0;

}

Input:-

Provided in image given below.

Output:

For Translation:

t.object();

break;

default:

cout<<"\nInvalid choice";

}

getch();

return 0;

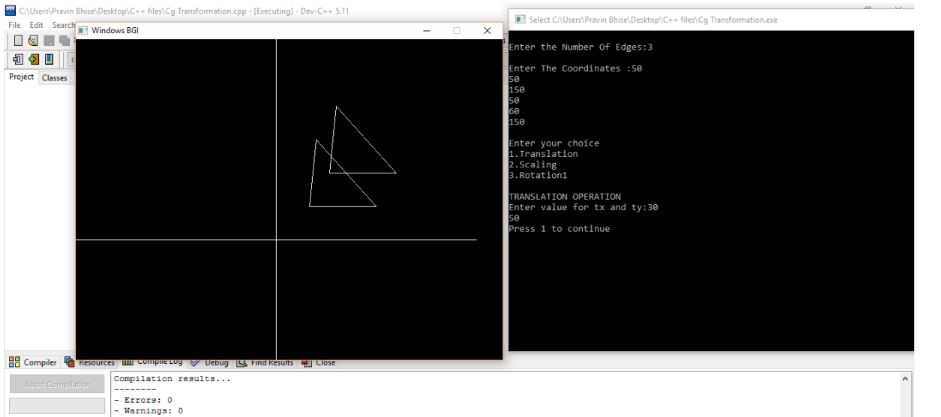
}

**Input**:-

Provided in image given below.

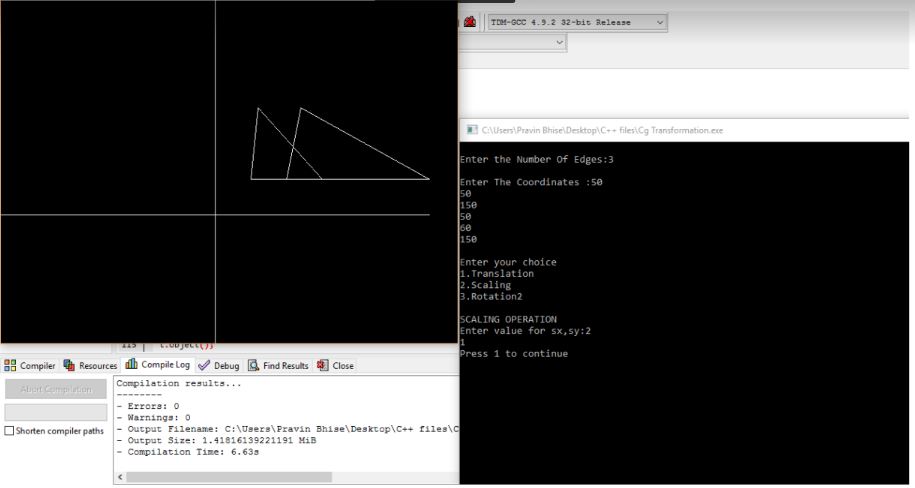
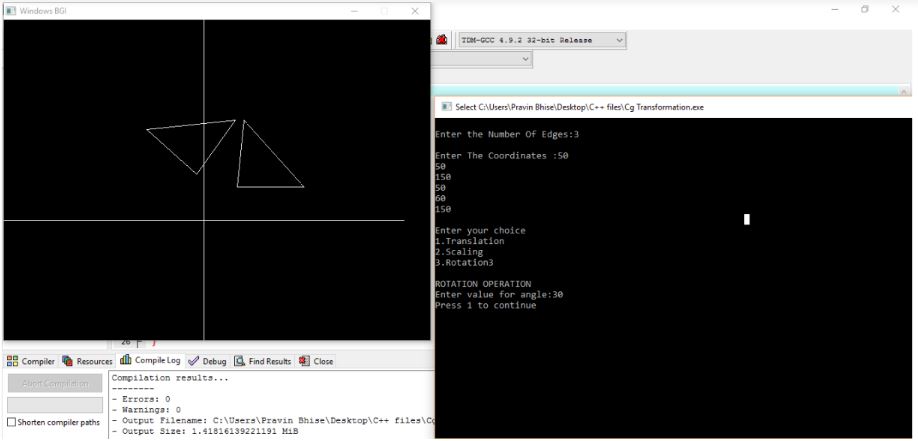
**Output**:

For Translation:



For Scaling :

For Rotation :



**Name** – Tejas Pravin Pawar ( SEB25 )

**Assignment no – 5**

**Title :** Write C++ Program To Generate Fractal Patterns By Using

Koch Curves

**Source Code :-**

#include <iostream>

#include <math.h>

#include <graphics.h>

using namespace std;

class kochCurve

{

public:

void koch(int it,int x1,int y1,int x5,int y5)

{

int x2,y2,x3,y3,x4,y4;

int dx,dy;

if (it==0)

{

line(x1,y1,x5,y5);

}

else

{

delay(10);

dx=(x5-x1)/3;

dy=(y5-y1)/3;

x2=x1+dx;

y2=y1+dy;

x3=(int)(0.5\*(x1+x5)+sqrt(3)\*(y1-y5)/6);

y3=(int)(0.5\*(y1+y5)+sqrt(3)\*(x5-x1)/6);

x4=2\*dx+x1;

y4=2\*dy+y1;

koch(it-1,x1,y1,x2,y2);

koch(it-1,x2,y2,x3,y3);

koch(it-1,x3,y3,x4,y4);

koch(it-1,x4,y4,x5,y5);

}

}

};

int main()

{

kochCurve k;

int it;

cout<<"Enter Number Of Iterations : "<<endl;

cin>>it;

int gd=DETECT,gm;

initgraph(&gd,&gm,NULL);

k.koch(it,150,20,20,280);

k.koch(it,280,280,150,20);

k.koch(it,20,280,280,280);

getch();

closegraph();

return 0;

}

int main()

{

kochCurve k;

int it;

cout<<"Enter Number Of Iterations : "<<endl;

cin>>it;

int gd=DETECT,gm;

initgraph(&gd,&gm,NULL);

k.koch(it,150,20,20,280);

k.koch(it,280,280,150,20);

k.koch(it,20,280,280,280);

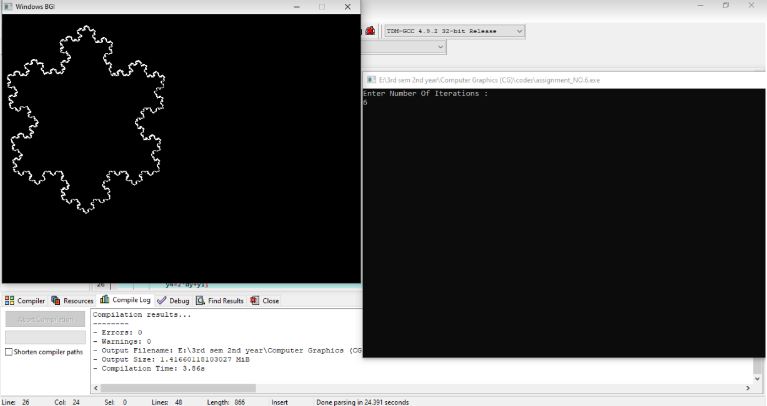
getch();

closegraph();

return 0;

}

**Output :**



**Name** – Tejas Pravin Pawar ( SEB25 )

**Assignment no – 7**

**Title :** Write a Program To Implement The Game Tic

Tac Toe. Apply The Concept of Polymorphism.

**Source Code :-**

//Tic Tac Toe Game in C++

//Importing the inbuild libraries in CPP

#include <iostream>

#include <stdlib.h>

using namespace std;

//Array for the board

char board[3][3] = {{'1','2','3'},{'4','5','6'},{'7','8','9'}};

//Variable Declaration

int choice;

int row,column;

char turn = 'X';

bool draw = false;

//Function to show the current status of the gaming board

void display\_board(){

//Rander Game Board LAYOUT

cout<<"PLAYER - 1 [X]t PLAYER - 2 [O]nn";

cout<<"tt | | n";

cout<<"tt "<<board[0][0]<<" | "<<board[0][1]<<" | "<<board[0][2]<<" n";

cout<<"tt\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_n";

cout<<"tt | | n";

cout<<"tt "<<board[1][0]<<" | "<<board[1][1]<<" | "<<board[1][2]<<" n";

cout<<"tt\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_n";

cout<<"tt | | n";

cout<<"tt "<<board[2][0]<<" | "<<board[2][1]<<" | "<<board[2][2]<<" n";

cout<<"tt | | n";

}

//Function to get the player input and update the board

void player\_turn(){

if(turn == 'X'){

cout<<"ntPlayer - 1 [X] turn : ";

}

else if(turn == 'O'){

cout<<"ntPlayer - 2 [O] turn : ";

}

//Taking input from user

//updating the board according to choice and reassigning the turn Start

cin>> choice;

//switch case to get which row and column will be update

switch(choice){

case 1: row=0; column=0; break;

case 2: row=0; column=1; break;

case 3: row=0; column=2; break;

case 4: row=1; column=0; break;

case 5: row=1; column=1; break;

case 6: row=1; column=2; break;

case 7: row=2; column=0; break;

case 8: row=2; column=1; break;

case 9: row=2; column=2; break;

default:

cout<<"Invalid Move";

}

if(turn == 'X' && board[row][column] != 'X' && board[row][column] != 'O'){

//updating the position for 'X' symbol if

//it is not already occupied

board[row][column] = 'X';

turn = 'O';

}else if(turn == 'O' && board[row][column] != 'X' && board[row][column] != 'O'){

//updating the position for 'O' symbol if

//it is not already occupied

board[row][column] = 'O';

turn = 'X';

}else {

//if input position already filled

cout<<"Box already filled!n Please choose another!!nn";

player\_turn();

}

/\* Ends \*/

display\_board();

}

//Function to get the game status e.g. GAME WON, GAME DRAW GAME IN CONTINUE MODE

bool gameover(){

//checking the win for Simple Rows and Simple Column

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

if(board[i][0] == board[i][1] && board[i][0] == board[i][2] || board[0][i] ==

board[1][i]

&& board[0][i] == board[2][i])

return false;

//checking the win for both diagonal

if(board[0][0] == board[1][1] && board[0][0] == board[2][2] || board[0][2] ==

board[1][1]

&& board[0][2] == board[2][0])

return false;

//Checking the game is in continue mode or not

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

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

if(board[i][j] != 'X' && board[i][j] != 'O')

return true;

//Checking the if game already draw

draw = true;

return false;

}

//Program Main Method

int main()

{

cout<<"tttT I C K -- T A C -- T O E -- G A M Ettt";

cout<<"nttttFOR 2 PLAYERSnttt";

while(gameover()){

display\_board();

player\_turn();

gameover();

}

if(turn == 'X' && draw == false){

cout<<"nnCongratulations!Player with 'X' has won the game";

}

else if(turn == 'O' && draw == false){

cout<<"nnCongratulations!Player with 'O' has won the game";

}

else

cout<<"nnGAME DRAW!!!nn";

}

cout<<"ntPlayer - 2 [O] turn : ";

}

//Taking input from user

//updating the board according to choice and reassigning the turn Start

cin>> choice;

//switch case to get which row and column will be update

switch(choice){

case 1: row=0; column=0; break;

case 2: row=0; column=1; break;

case 3: row=0; column=2; break;

case 4: row=1; column=0; break;

case 5: row=1; column=1; break;

case 6: row=1; column=2; break;

case 7: row=2; column=0; break;

case 8: row=2; column=1; break;

case 9: row=2; column=2; break;

default:

cout<<"Invalid Move";

}

if(turn == 'X' && board[row][column] != 'X' && board[row][column] != 'O'){

//updating the position for 'X' symbol if

//it is not already occupied

board[row][column] = 'X';

turn = 'O';

}else if(turn == 'O' && board[row][column] != 'X' && board[row][column] != 'O'){

//updating the position for 'O' symbol if

//it is not already occupied

board[row][column] = 'O';

turn = 'X';

}else {

//if input position already filled

cout<<"Box already filled!n Please choose another!!nn";

player\_turn();

}

/\* Ends \*/

display\_board();

}

//Function to get the game status e.g. GAME WON, GAME DRAW GAME IN CONTINUE MODE

bool gameover(){

//checking the win for Simple Rows and Simple Column

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

if(board[i][0] == board[i][1] && board[i][0] == board[i][2] || board[0][i] ==

board[1][i]

&& board[0][i] == board[2][i])

return false;

//checking the win for both diagonal

if(board[0][0] == board[1][1] && board[0][0] == board[2][2] || board[0][2] ==

board[1][1]

&& board[0][2] == board[2][0])

return false;

//Checking the game is in continue mode or not

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

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

if(board[i][j] != 'X' && board[i][j] != 'O')

return true;

//Checking the if game already draw

draw = true;

return false;

}

//Program Main Method

int main()

{

cout<<"tttT I C K -- T A C -- T O E -- G A M Ettt";

cout<<"nttttFOR 2 PLAYERSnttt";

while(gameover()){

display\_board();

player\_turn();

gameover();

}

if(turn == 'X' && draw == false){

cout<<"nnCongratulations!Player with 'X' has won the game";

}

else if(turn == 'O' && draw == false){

cout<<"nnCongratulations!Player with 'O' has won the game";

}

else

cout<<"nnGAME DRAW!!!nn";

}

//checking the win for Simple Rows and Simple Column

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

if(board[i][0] == board[i][1] && board[i][0] == board[i][2] || board[0][i] ==

board[1][i]

&& board[0][i] == board[2][i])

return false;

//checking the win for both diagonal

if(board[0][0] == board[1][1] && board[0][0] == board[2][2] || board[0][2] ==

board[1][1]

&& board[0][2] == board[2][0])

return false;

//Checking the game is in continue mode or not

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

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

if(board[i][j] != 'X' && board[i][j] != 'O')

return true;

//Checking the if game already draw

draw = true;

return false;

}

//Program Main Method

int main()

{

cout<<"tttT I C K -- T A C -- T O E -- G A M Ettt";

cout<<"nttttFOR 2 PLAYERSnttt";

while(gameover()){

display\_board();

player\_turn();

gameover();

}

if(turn == 'X' && draw == false){

cout<<"nnCongratulations!Player with 'X' has won the game";

}

else if(turn == 'O' && draw == false){

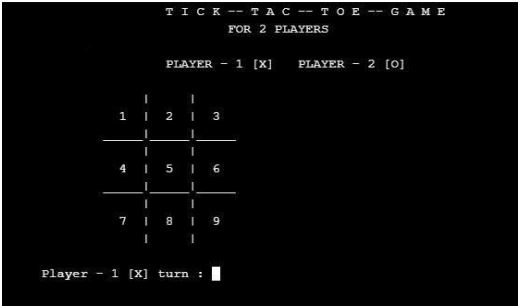
cout<<"nnCongratulations!Player with 'O' has won the game";

}

else

cout<<"nnGAME DRAW!!!nn";

}



**Output :**