**Assignment No.:- 01**

Program:- write a c++ program to draw a concave polygon and fill it with desired colour using scan fillalgorithm.

Name:- Tejaswini Sachin Shewale

Roll:- 37

#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);

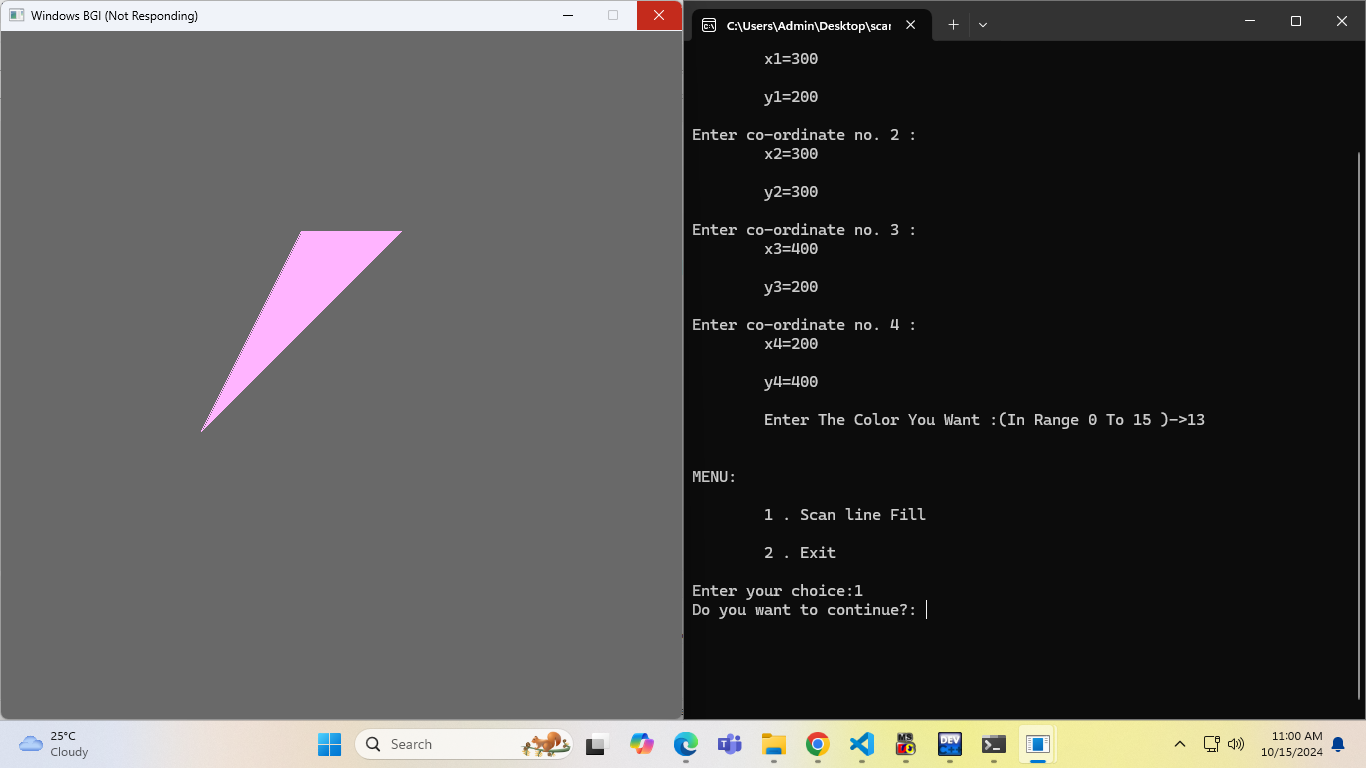
x.display();

closegraph(); //closing graph

getch();

return 0;

}

Output: 

**Assignment No:- 02**

Program:- write a c++ program to implement cohen southerland line clipping algorithm.

Name:- Tejaswini Sachin Shewale

Roll:- 37

#include<iostream>

#include<graphics.h>

typedef unsigned int outcode;

enum{TOP=0x1,BOTTOM=0x2,RIGHT=0x4,LEFT=0x8};

using namespace std;

outcode CompOutCode(double ,double ,double ,double ,double ,double );

void CSLCAD(double x0,double y0,double x1,double y1,double xmin,double xmax,double ymin,double ymax)

{

outcode outcode0,outcode1,outcodeout;

boolean accept=FALSE, done=FALSE;

outcode0=CompOutCode(x0,y0,xmin,xmax,ymin,ymax);

outcode1=CompOutCode(x1,y1,xmin,xmax,ymin,ymax);

cout<<"outcode0="<<outcode0<<endl;

cout<<"outcode1="<<outcode1<<endl;

do

{

if(outcode0==0 && outcode1==0)

{

accept=TRUE;

done=TRUE;

}

else if(outcode0 & outcode1)

{

done=TRUE;

}

else

{

double x,y;

int ocd=outcode0 ? outcode0:outcode1;

if(ocd & TOP)

{

x=x0+(x1-x0)\*(ymax-y0)/(y1-y0);

y=ymax;

}

else if(ocd & BOTTOM)

{

x=x0+(x1-x0)\*(ymin-y0)/(y1-y0);

y=ymin;

}

else if(ocd & LEFT)

{

y=y0+(y1-y0)\*(xmin-x0)/(x1-x0);

x=xmin;

} else

{

y=y0+(y1-y0)\*(xmax-x0)/(x1-x0);

x=xmax;

}

if(ocd==outcode0)

{

x0=x;

y0=y;

outcode0=CompOutCode(x0,y0,xmin,xmax,ymin,ymax);

} else

{

x1=x;

y1=y;

outcode1=CompOutCode(x1,y1,xmin,xmax,ymin,ymax);

}

}

}while(done==FALSE);

if(accept==TRUE)

{

line(x0,y0,x1,y1);

}

} outcode CompOutCode(double x,double y,double xmin,double xmax,double ymin,double ymax)

{

outcode code=0;

if(y>ymax)

code|=TOP;

if(y<ymin)

code|=BOTTOM;

if(x>xmax)

code|=RIGHT;

if(x<xmin)

code|=LEFT;

return code;

} int main()

{

string ch;

double xmin,xmax,ymin,ymax,x0,y0,x1,y1;

initwindow(500,600);

cout<<"Enter the bottom co-ordinates of window:";

cin>>xmin;

cout<<"Enter the left coordinates of the window:";

cin>>ymin;

cout<<"Enter the right coordinates of the window:";

cin>>xmax;

cout<<"Enter the top coordinates of the window:";

cin>>ymax;

rectangle(xmin,ymin,xmax,ymax);

cout<<"Enter the coordinates(Terminal Points) of the line: ";

cin>>x0>>y0;

cin>>x1>>y1;

line(x0,y0,x1,y1);

delay(5000);

cleardevice();

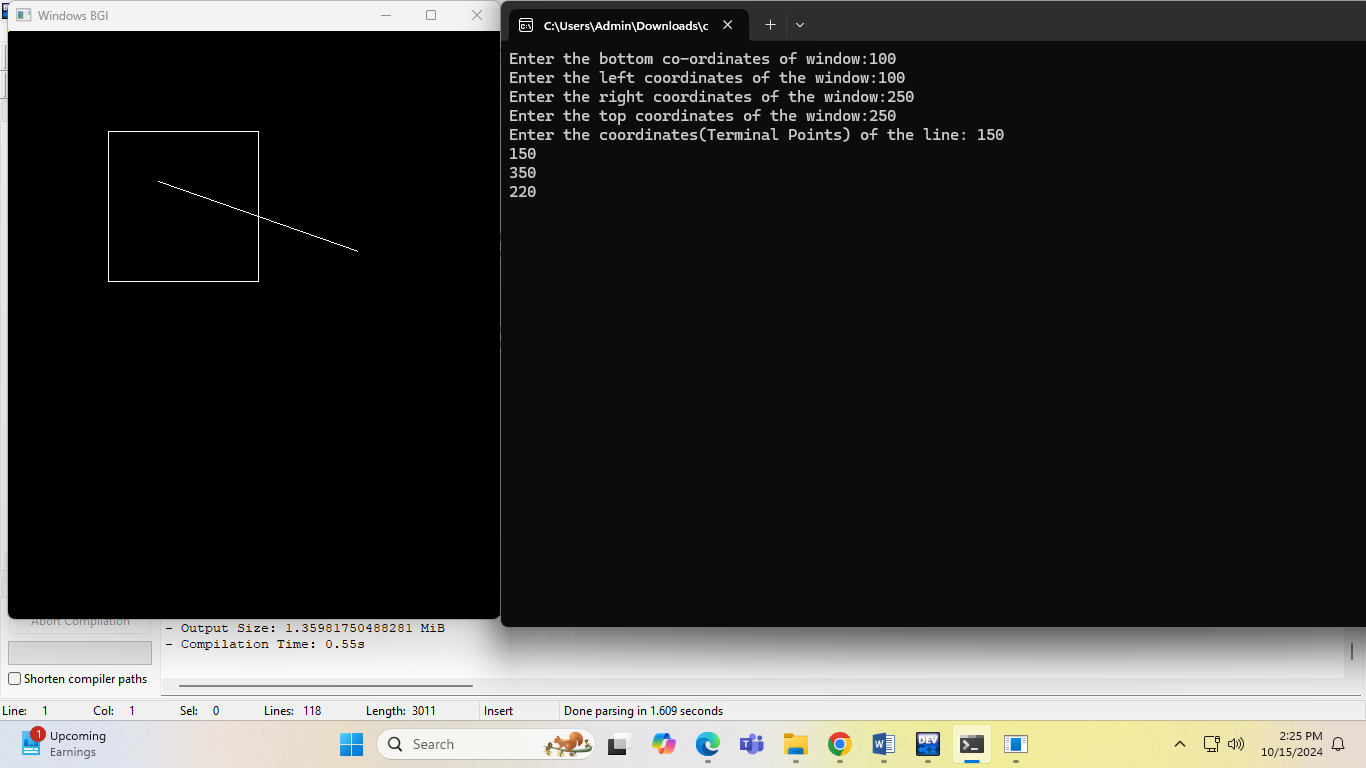
CSLCAD(x0,y0,x1,y1,xmin,xmax,ymin,ymax);

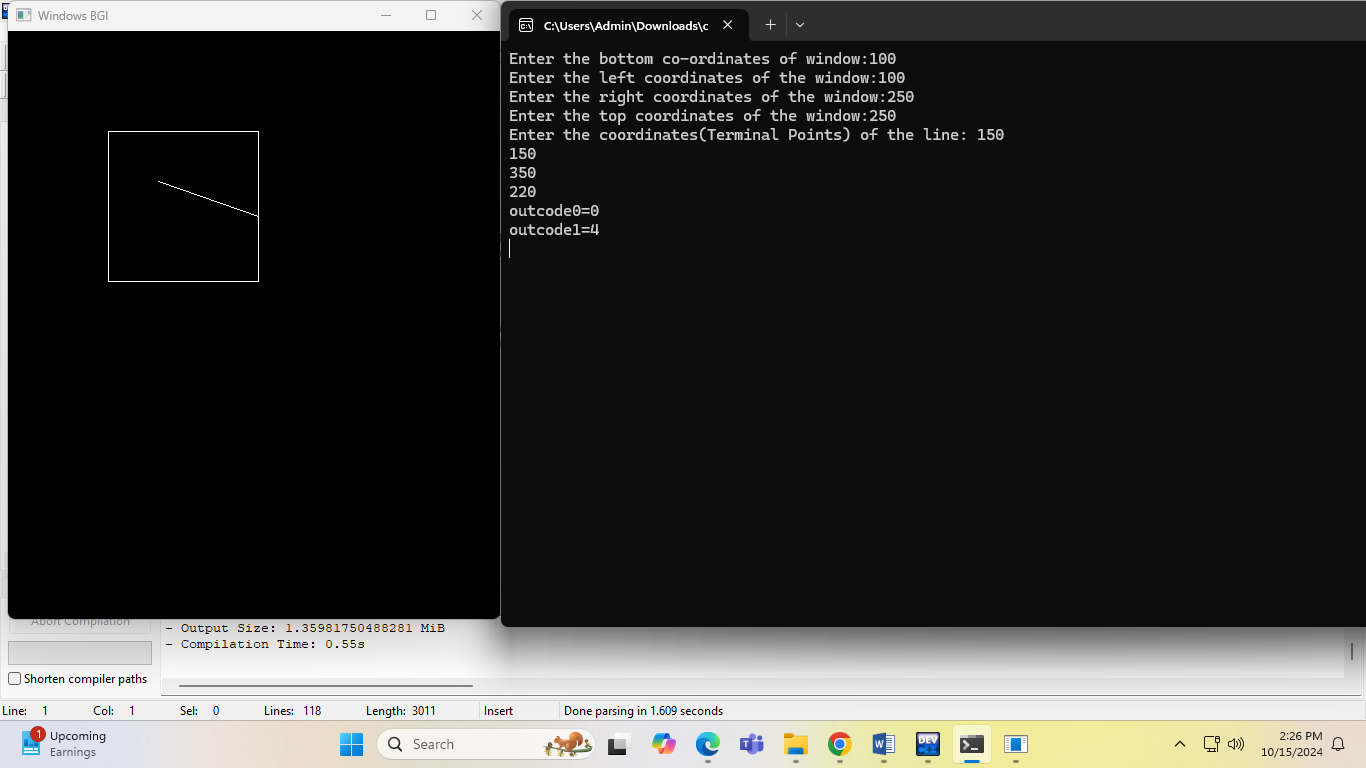
rectangle(xmin,ymin,xmax,ymax);

delay(50000);

closegraph(); }

Output:





**Assiginment No:03**

Program:- write a c++ program to draw the pattern.use DDA line and bresenham’s circle drawing algorithm.apply the concept of encapsulation.

Name:- Tejaswini Sachin Shewale

Roll:- 37

#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 << "\nEnter 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;

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

cin >> n;

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

cout << "Enter coordinates of point x1 : ";

cin >> x1;

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

cin >> y1;

cout << "Enter coordinates 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 << "\nEnter 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();

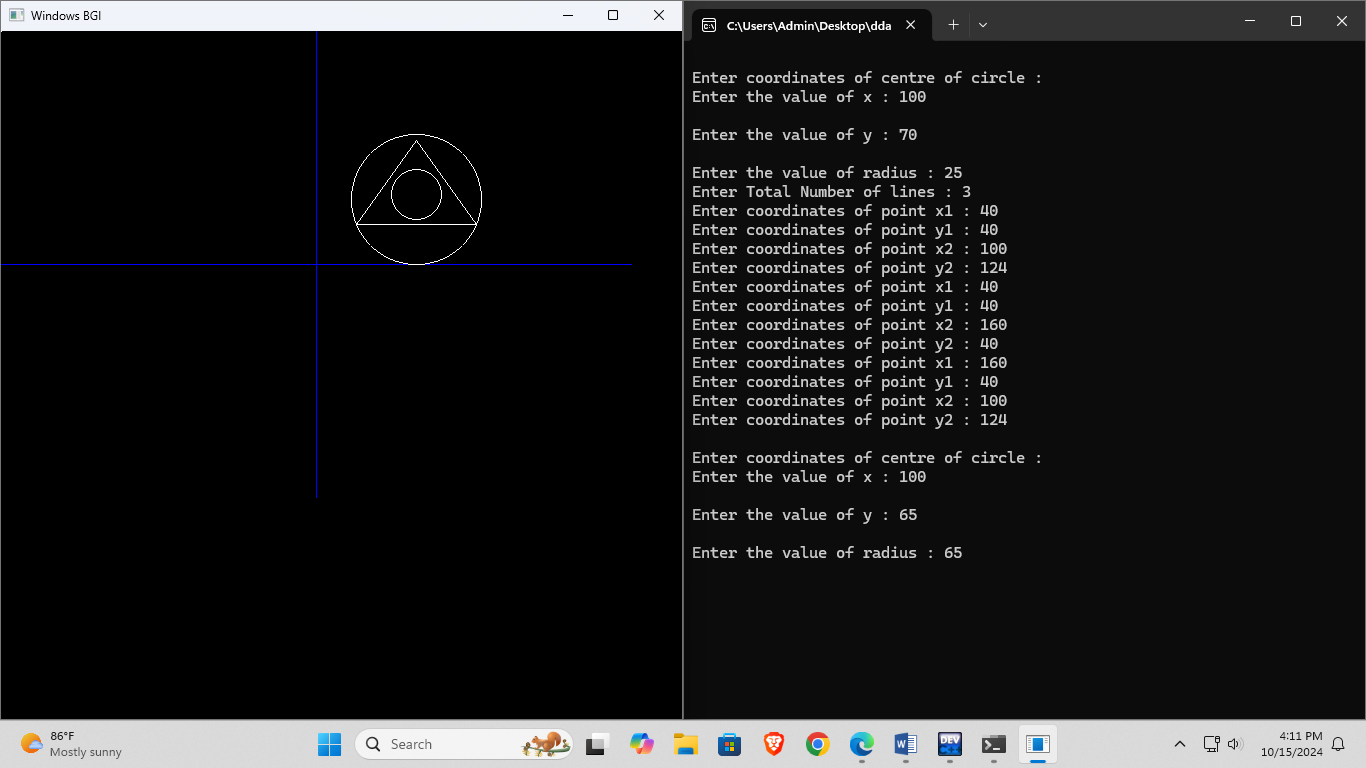
delay(200);

closegraph();

return 0;

}

Output:



**Assignment No:- 04**

Program:- write a c++ program to draw 2-D object and perform the following basic transformation:

1.scaling

2.Translation

3.Rotation. Apply the concept of operator overloading

Name:- Tejaswini Sachin Shewale

Roll:- 37

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:

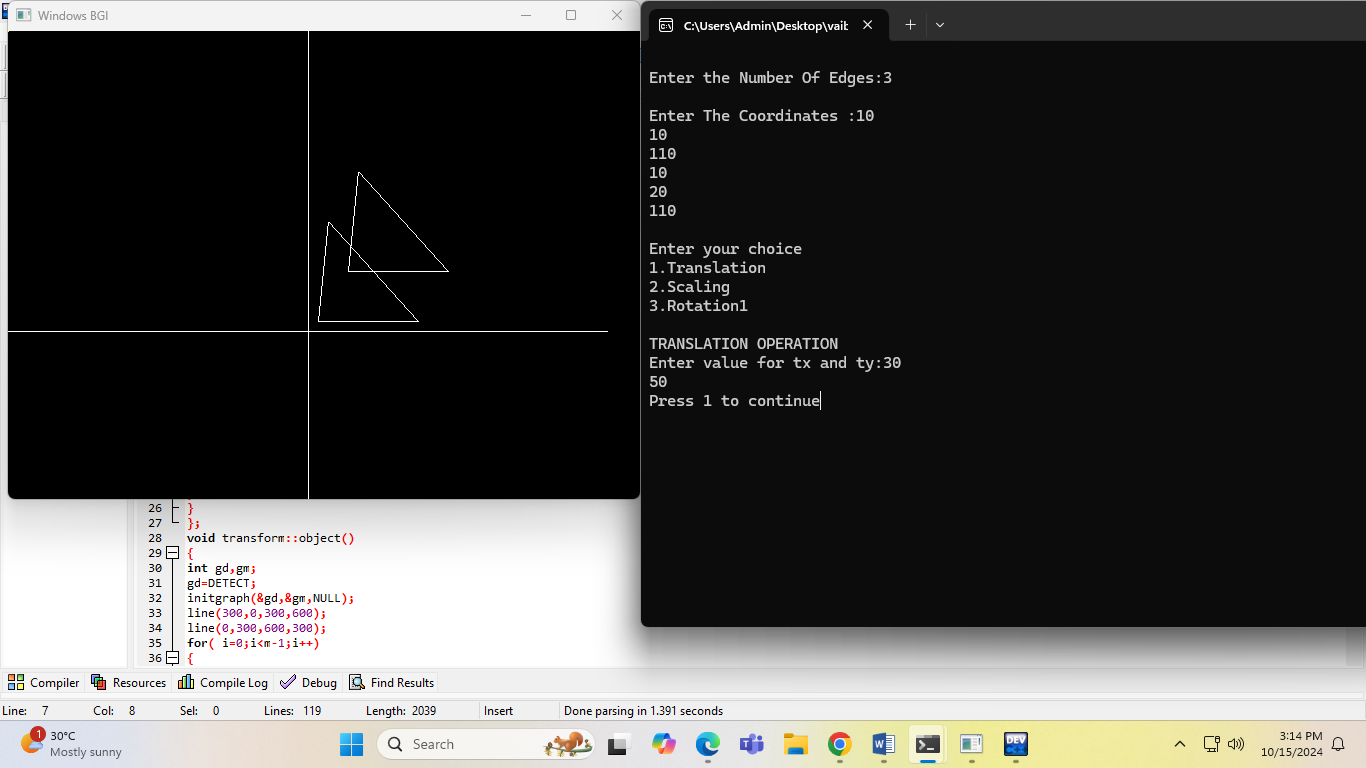
cout<<"\nInvalid choice";

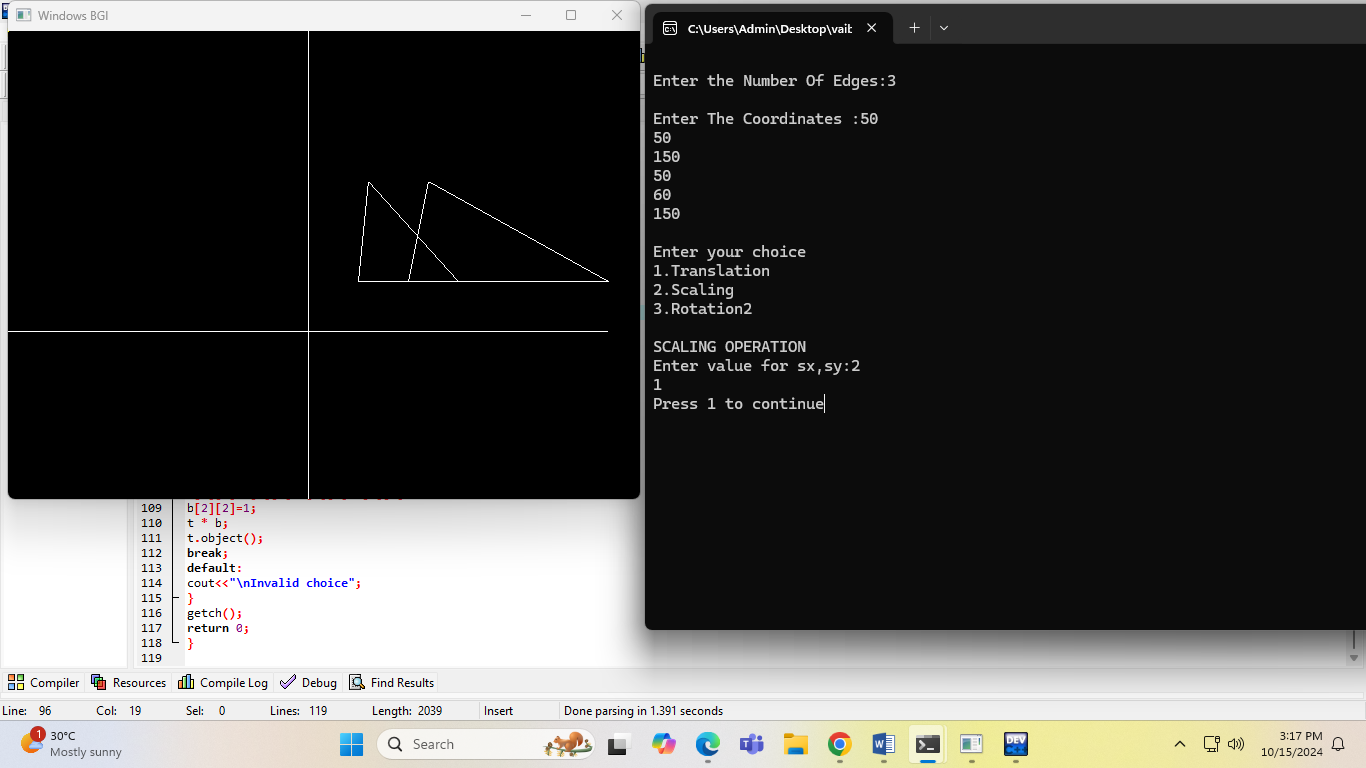
}getch();

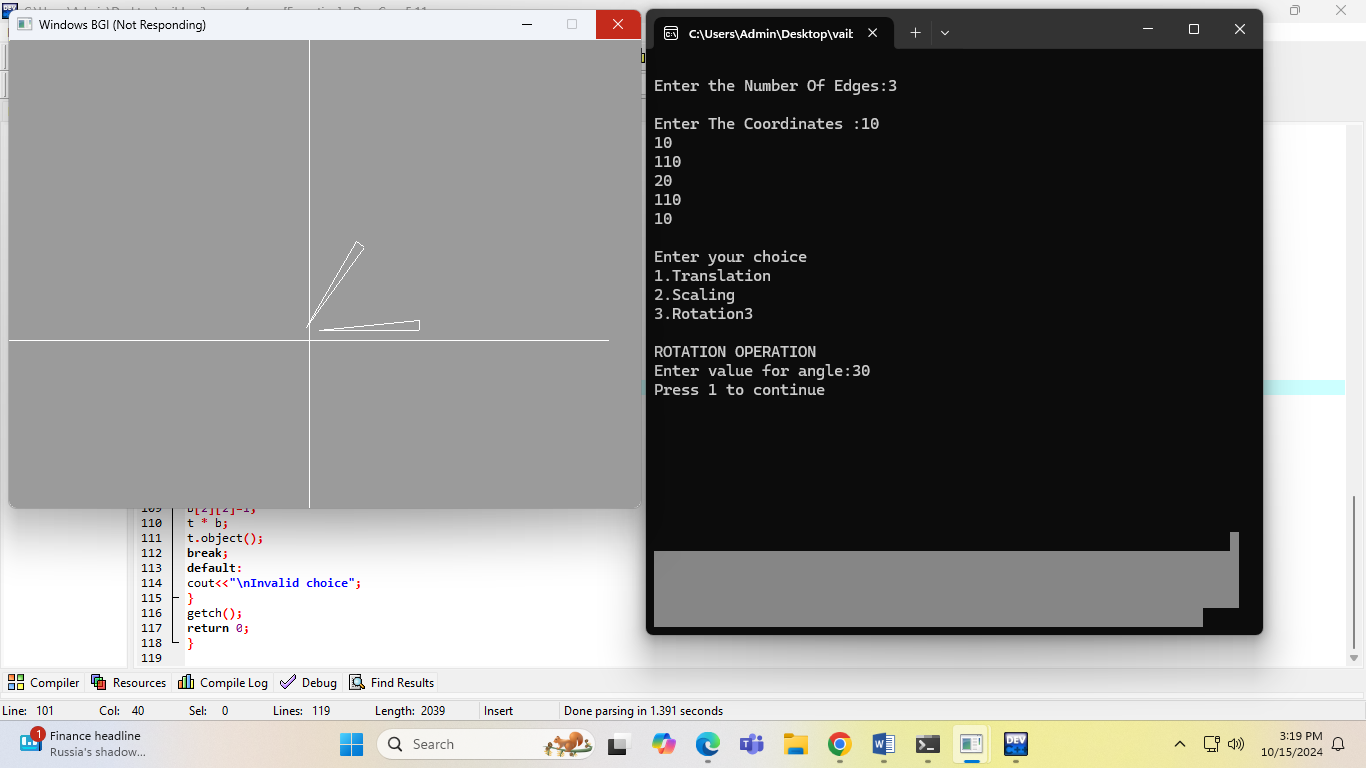
return 0;

}

Output:







**Assignment No:- 05**

Program:-Write a c++ program to generate a Hilbert curve using the concept of fractals.

Name:-Tejaswini Sachin Shewale

Roll:- 37

**Code:**

#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<<"Enter the value to draw the number of curves=";

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);

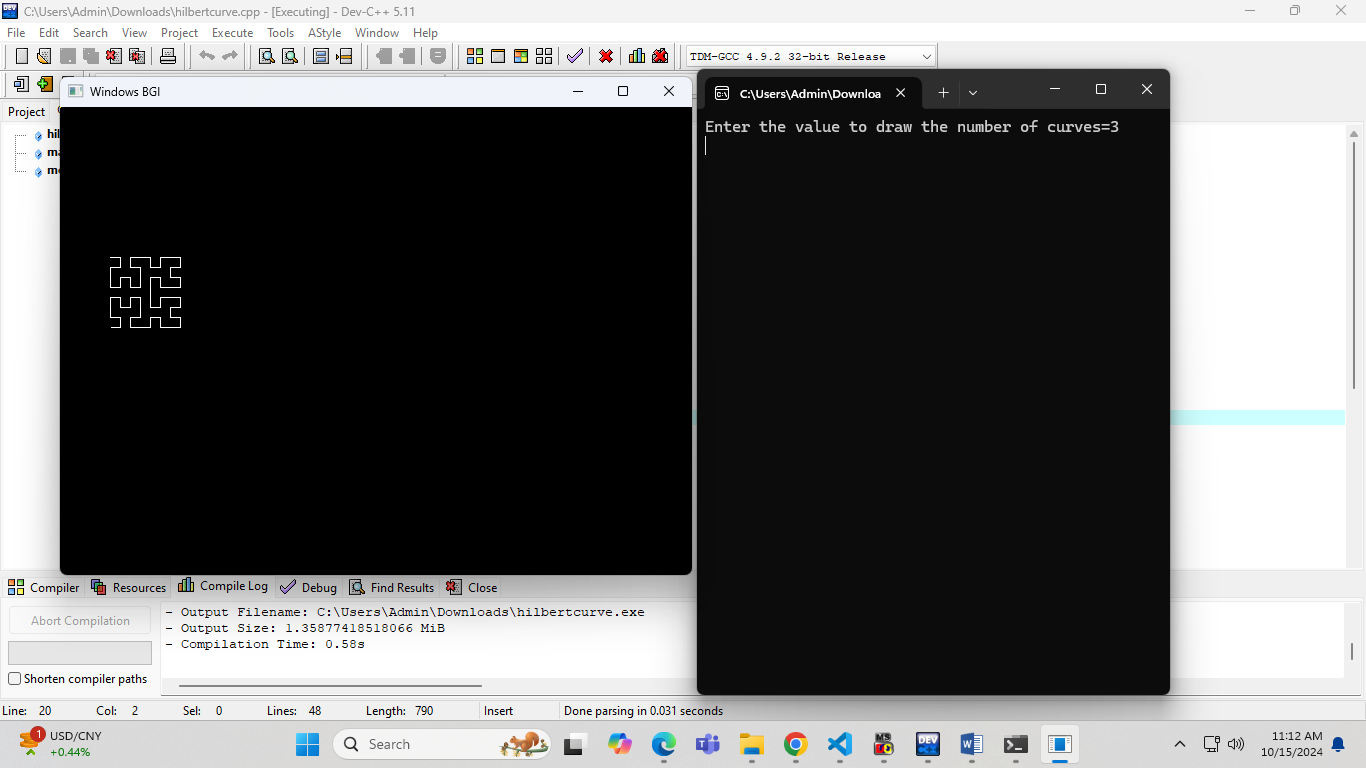
delay(10000);

closegraph();

return 0;

}

Output:



Assignment No:- 06

Program:-write a c++ program to draw man walking in the rain with an umbrella.apply the concept of polymorphism.

Name:- Tejaswini Sachin Shewale

Roll:- 37

#include <iostream>

#include <conio.h>

#include <graphics.h>

#include <stdlib.h>

#include <dos.h>

using namespace std;

class walkingman {

int rhx, rhy;

public:

void draw(int, int);

void draw(int);

};

void walkingman::draw(int i) {

setcolor(WHITE);

line(20, 380, 580, 380);

setcolor(LIGHTBLUE);

if (i % 2) {

setcolor(RED);

line(25 + i, 380, 35 + i, 340);

line(45 + i, 380, 35 + i, 340);

line(35 + i, 310, 25 + i, 330);

delay(20);

} else {

setcolor(GREEN);

line(35 + i, 340, 35 + i, 310);

line(35 + i, 310, 40 + i, 330);

delay(20);

}

setcolor(YELLOW);

line(35 + i, 340, 35 + i, 310);

setcolor(RED);

circle(35 + i, 300, 10);

setcolor(CYAN);

line(35 + i, 310, 50 + i, 330);

setcolor(MAGENTA);

line(50 + i, 330, 50 + i, 280);

line(15 + i, 280, 85 + i, 280);

setcolor(WHITE);

arc(50 + i, 280, 0, 180, 35);

arc(55 + i, 330, 180, 360, 5);

}

void walkingman::draw(int x, int y) {

int j;

rhx = x;

rhy = y;

for (j = 0; j < 100; j++) {

setcolor(WHITE);

outtextxy(rand() % rhx, rand() % (rhy - 50), "|");

}

}

int main() {

int gd = DETECT, gm;

int rhx, rhy, j, i;

walkingman obj;

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

for (i = 0; i < 500; i++) {

obj.draw(i);

rhx = getmaxx();

rhy = getmaxy();

obj.draw(rhx, rhy);

delay(150);

cleardevice();

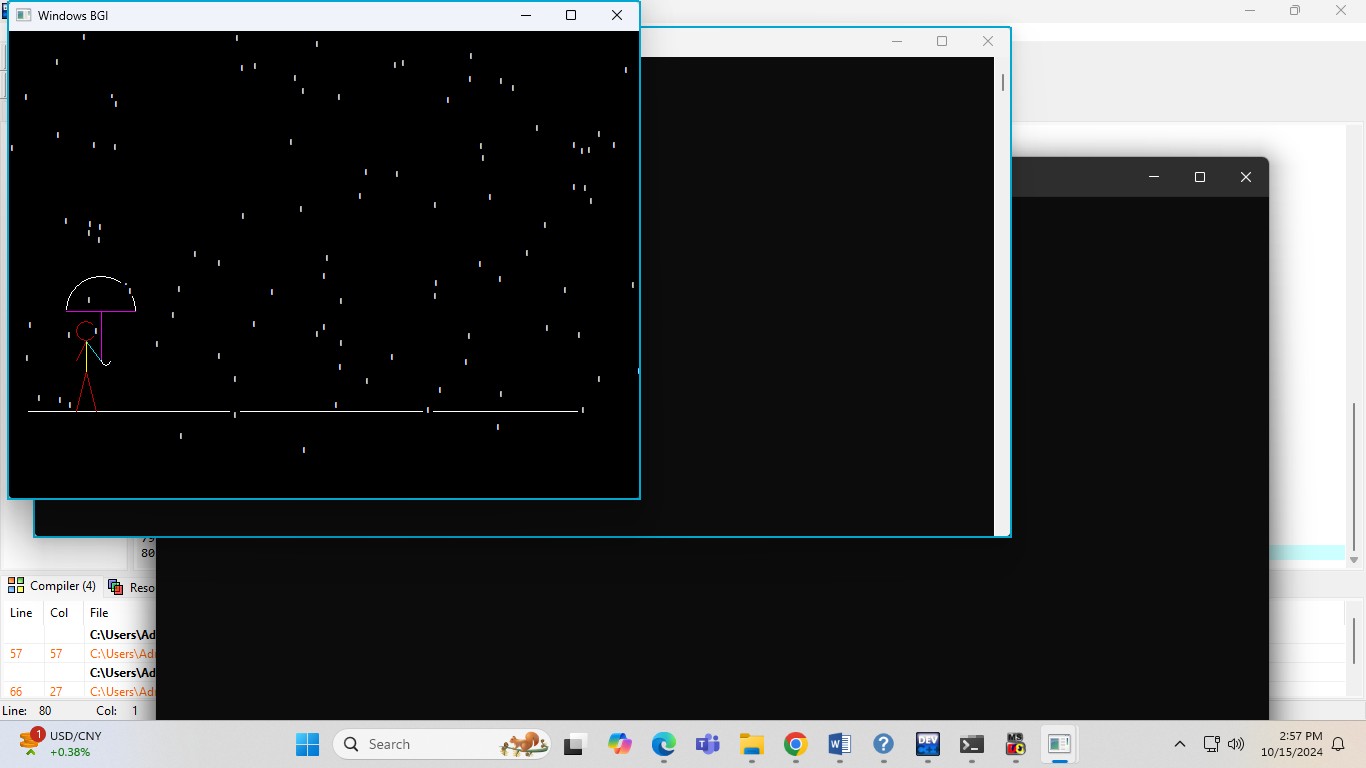
} getch();

closegraph();

return 0;

}

Output:



**Assignment No:- 07**

Program:- write a c++ program to draw 3-D cube and perform following transformation on it usingopengl.

1)scaling 2)Translation 3)Rotation about an axis(X/Y/Z).

Name:- Tejaswini Sachin Shewale

Roll:- 37

#include <iostream>

#include <GL/glut.h>

#include <cmath>

using namespace std;

// Global variables for transformations

3)Rotation about an axis(X/Y/Z).

float angleX = 0.0f, angleY = 0.0f, angleZ = 0.0f;

float translateX = 0.0f, translateY = 0.0f, translateZ = 0.0f;

float scaleX = 1.0f, scaleY = 1.0f, scaleZ = 1.0f;

int choice;

// Function to draw a cube

void drawCube() {

glBegin(GL\_QUADS);

// Front face (z = 1)

glColor3f(1.0f, 0.0f, 0.0f); // Red

glVertex3f(-1.0f, -1.0f, 1.0f);

glVertex3f(1.0f, -1.0f, 1.0f);

glVertex3f(1.0f, 1.0f, 1.0f);

glVertex3f(-1.0f, 1.0f, 1.0f);

// Back face (z = -1)

glColor3f(0.0f, 1.0f, 0.0f); // Green

glVertex3f(-1.0f, -1.0f, -1.0f);

glVertex3f(-1.0f, 1.0f, -1.0f);

glVertex3f(1.0f, 1.0f, -1.0f);

glVertex3f(1.0f, -1.0f, -1.0f);

// Left face (x = -1)

glColor3f(0.0f, 0.0f, 1.0f); // Blue

glVertex3f(-1.0f, -1.0f, -1.0f);

glVertex3f(-1.0f, -1.0f, 1.0f);

glVertex3f(-1.0f, 1.0f, 1.0f);

glVertex3f(-1.0f, 1.0f, -1.0f);

// Right face (x = 1)

glColor3f(1.0f, 1.0f, 0.0f); // Yellow

glVertex3f(1.0f, -1.0f, -1.0f);

glVertex3f(1.0f, 1.0f, -1.0f);

glVertex3f(1.0f, 1.0f, 1.0f);

glVertex3f(1.0f, -1.0f, 1.0f);

// Top face (y = 1)

glColor3f(0.0f, 1.0f, 1.0f); // Cyan

glVertex3f(-1.0f, 1.0f, -1.0f);

glVertex3f(-1.0f, 1.0f, 1.0f);

glVertex3f(1.0f, 1.0f, 1.0f);

glVertex3f(1.0f, 1.0f, -1.0f);

// Bottom face (y = -1)

glColor3f(1.0f, 0.0f, 1.0f); // Magenta

glVertex3f(-1.0f, -1.0f, -1.0f);

glVertex3f(1.0f, -1.0f, -1.0f);

glVertex3f(1.0f, -1.0f, 1.0f);

glVertex3f(-1.0f, -1.0f, 1.0f);

glEnd();

}

// Display callback function

void display() {

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

glLoadIdentity();

// Apply transformations

glTranslatef(translateX, translateY, translateZ);

glScalef(scaleX, scaleY, scaleZ);

glRotatef(angleX, 1.0f, 0.0f, 0.0f);

glRotatef(angleY, 0.0f, 1.0f, 0.0f);

glRotatef(angleZ, 0.0f, 0.0f, 1.0f);

drawCube();

glutSwapBuffers();

}

// Initialization function

void init() {

glEnable(GL\_DEPTH\_TEST); // Enable depth testing

glClearColor(1.0f, 1.0f, 1.0f, 1.0f); // Set background color to white

}

// Reshape callback function

void reshape(int width, int height) {

glViewport(0, 0, width, height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(45.0f, (float)width / (float)height, 0.1f, 100.0f);

glMatrixMode(GL\_MODELVIEW);

}

// Main function

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

glutInit(&argc, argv);

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

glutInitWindowSize(800, 600);

glutInitWindowPosition(100, 100);

glutCreateWindow("3D Cube Transformations");

init();

glutDisplayFunc(display);

glutReshapeFunc(reshape);

// User input for transformations

cout << "Choose a transformation:\n";

cout << "1. Translation\n";

cout << "2. Scaling\n";

cout << "3. Rotation\n";

cout << "=> ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter translation values (Tx, Ty, Tz): ";

cin >> translateX >> translateY >> translateZ;

break;

case 2:

cout << "Enter scaling values (Sx, Sy, Sz): ";

cin >> scaleX >> scaleY >> scaleZ;

break;

case 3:

cout << "Enter rotation angles (angleX, angleY, angleZ): ";

cin >> angleX >> angleY >> angleZ;

break;

default:

cout << "Invalid choice!\n";

return 1;

}

glutMainLoop();

return 0;

}

Output: 