**UCS 1712 – GRAPHICS AND MULTIMEDIA LAB**

**ASSIGNMENT – 5**

**VISHAL N**

**185001198**

* + 1. **CSEC**

**1. 2-D TRANSFORMATIONS IN C++ USING OPENGL:**

#include <GL/glut.h>

// #include <GL/freeglut.h>

#include <stdio.h>

#include <iostream>

#include <math.h>

#include <vector>

#include <cmath>

using namespace std;

vector<int> x\_coordinates;

vector<int> y\_coordinates;

int x\_trans, y\_trans, x\_fixed, y\_fixed, x\_scale, y\_scale, x\_shear, y\_shear;

double rotation\_angle;

int polygon, edges, choice, x, y;

void myInit(void) {

    glClearColor(1.0, 1.0, 1.0, 0.0);

    glColor3f(0.0f, 0.0f, 0.0f);

    glPointSize(0.05);

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    gluOrtho2D(-600.0, 600.0, -600.0, 600.0);

}

void drawAxis(void){

    glBegin(GL\_LINES);

    glVertex2d(-600, 0);

    glVertex2d(600, 0);

    glEnd();

    glBegin(GL\_LINES);

    glVertex2d(0, -600);

    glVertex2d(0, 600);

    glEnd();

}

double round(double d)

{

    return floor(d + 0.5);

}

void drawPolygon(void){

    if (polygon == 1){

        glBegin(GL\_LINES);

    }

    else{

        glBegin(GL\_POLYGON);

    }

    for (int i = 0; i < edges; i++)

    {

        glVertex2d(x\_coordinates[i], y\_coordinates[i]);

    }

    glEnd();

}

void drawTranslatedPolygon(int x, int y){

    if (polygon == 1)

    {

        glBegin(GL\_LINES);

    }

    else{

        glBegin(GL\_POLYGON);

    }

    for (int i = 0; i < edges; i++)

    {

        glVertex2d(x\_coordinates[i] + x, y\_coordinates[i] + y);

    }

    glEnd();

}

void drawRotatedOriginPolygon(double rotation\_angle){

    if (polygon == 1)

    {

        glBegin(GL\_LINES);

    }

    else{

        glBegin(GL\_POLYGON);

    }

    for (int i = 0; i < edges; i++)

    {

        glVertex2d(round((x\_coordinates[i] \* cos(rotation\_angle)) - (y\_coordinates[i] \* sin(rotation\_angle))), round((x\_coordinates[i] \* sin(rotation\_angle)) + (y\_coordinates[i] \* cos(rotation\_angle))));

    }

    glEnd();

}

void drawRotatedFixedPointPolygon(int x, int y, double rotation\_angle){

    if (polygon == 1)

    {

        glBegin(GL\_LINES);

    }

    else{

        glBegin(GL\_POLYGON);

    }

    for (int i = 0; i < edges; i++)

    {

        x\_coordinates[i] += x;

        y\_coordinates[i] += y;

    }

    for (int i = 0; i < edges; i++)

    {

        glVertex2d(round((x\_coordinates[i] \* cos(rotation\_angle)) - (y\_coordinates[i] \* sin(rotation\_angle))), round((x\_coordinates[i] \* sin(rotation\_angle)) + (y\_coordinates[i] \* cos(rotation\_angle))));

    }

    glEnd();

}

void drawScaledOriginPolygon(double x, double y){

    if (polygon == 1)

    {

        glBegin(GL\_LINES);

    }

    else{

        glBegin(GL\_POLYGON);

    }

    for (int i = 0; i < edges; i++)

    {

        glVertex2d(round(x\_coordinates[i] \* x), round(y\_coordinates[i] \* y));

    }

    glEnd();

}

void drawScaledFixedPointPolygon(double x, double y, int x\_trans, int y\_trans){

    if (polygon == 1)

    {

        glBegin(GL\_LINES);

    }

    else

    {

        glBegin(GL\_POLYGON);

    }

    // To Scale around a fixed point, first translate(-x, -y), then scale(x, y), finally translate(x, y)

    for (int i = 0; i < edges; i++)

    {

        glVertex2d(x\_coordinates[i] - x, y\_coordinates[i] - y);

    }

    for (int i = 0; i < edges; i++)

    {

        glVertex2d(round(x\_coordinates[i] \* x), round(y\_coordinates[i] \* y));

    }

    for (int i = 0; i < edges; i++)

    {

        glVertex2d(x\_coordinates[i] + x, y\_coordinates[i] + y);

    }

    glEnd();

}

void drawReflectionXPolygon(){

    if (polygon == 1)

    {

        glBegin(GL\_LINES);

    }

    else

    {

        glBegin(GL\_POLYGON);

    }

    for (int i = 0; i < edges; i++)

    {

        glVertex2d(x\_coordinates[i] , y\_coordinates[i] \* -1);

    }

    glEnd();

}

void drawReflectionYPolygon(){

    if (polygon == 1)

    {

        glBegin(GL\_LINES);

    }

    else

    {

        glBegin(GL\_POLYGON);

    }

    for (int i = 0; i < edges; i++)

    {

        glVertex2d(x\_coordinates[i] \* -1 , y\_coordinates[i]);

    }

    glEnd();

}

void drawReflectionOriginPolygon(){

    if (polygon == 1)

    {

        glBegin(GL\_LINES);

    }

    else

    {

        glBegin(GL\_POLYGON);

    }

    for (int i = 0; i < edges; i++)

    {

        glVertex2d(x\_coordinates[i] \* -1 , y\_coordinates[i] \* -1);

    }

    glEnd();

}

void drawReflectionXequalsYPolygon(){

    if (polygon == 1)

    {

        glBegin(GL\_LINES);

    }

    else

    {

        glBegin(GL\_POLYGON);

    }

    for (int i = 0; i < edges; i++)

    {

        glVertex2d(y\_coordinates[i], x\_coordinates[i]);

    }

    glEnd();

}

void drawXShearedPolygon(int x\_shear){

    if (polygon == 1)

    {

        glBegin(GL\_LINES);

    }

    else

    {

        glBegin(GL\_POLYGON);

    }

    for (int i = 0; i < edges; i++)

    {

        glVertex2d(x\_coordinates[i] + (x\_shear \* y\_coordinates[i]), y\_coordinates[i]);

    }

    glEnd();

}

void drawYShearedPolygon(int y\_shear){

    if (polygon == 1)

    {

        glBegin(GL\_LINES);

    }

    else

    {

        glBegin(GL\_POLYGON);

    }

    for (int i = 0; i < edges; i++)

    {

        glVertex2d(x\_coordinates[i], y\_coordinates[i] + (y\_shear \* x\_coordinates[i]));

    }

    glEnd();

}

void myDisplay(void){

    glClear(GL\_COLOR\_BUFFER\_BIT);

    glColor3f(0.0, 0.0, 0.0);

    drawAxis();

    drawPolygon();

        glClear(GL\_COLOR\_BUFFER\_BIT);

        glColor3f(0.0, 0.0, 0.0);

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

        cout << "1. Translation" << endl;

        cout << "2. Rotation about Origin" << endl;

        cout << "3. Rotation wrt fixed point" << endl;

        cout << "4. Scaling wrt Origin" << endl;

        cout << "5. Scaling wrt fixed point" << endl;

        cout << "6. Reflection wrt x-axis" << endl;

        cout << "7. Reflection wrt y-axis" << endl;

        cout << "8. Reflection wrt origin" << endl;

        cout << "9. Reflection wrt line y=x" << endl;

        cout << "10. Shearing along x-direction" << endl;

        cout << "11. Shearing along y-direction" << endl;

        cout << "12. Exit" << endl;

        cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

        cin >> choice;

        if (choice == 12)

        {

            glFlush();

        }

        if (choice == 1)

        {

            cout << "Enter the translation factor for X and Y: ";

            cin >> x\_trans >> y\_trans;

            drawAxis();

            drawPolygon();

            drawTranslatedPolygon(x\_trans, y\_trans);

        }

        if (choice == 2)

        {

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

            cin >> rotation\_angle;

            rotation\_angle \*= M\_PI / 180;

            drawAxis();

            drawPolygon();

            drawRotatedOriginPolygon(rotation\_angle);

        }

        if (choice == 3)

        {

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

            cin >> rotation\_angle;

            rotation\_angle \*= M\_PI / 180;

            cout << "Enter the coordinates of the fixed point to rotate about: " << endl;

            cin >> x\_fixed >> y\_fixed;

            drawAxis();

            drawPolygon();

            drawRotatedFixedPointPolygon(x\_fixed, y\_fixed, rotation\_angle);

        }

        if (choice == 4)

        {

            cout << "Enter the scaling factor for X and Y: ";

            cin >> x\_scale >> y\_scale;

            drawAxis();

            drawPolygon();

            drawScaledOriginPolygon(x\_scale, y\_scale);

        }

        if (choice == 5)

        {

            cout << "Enter the scaling factor for X and Y and coordinates of the fixed point to scale about: ";

            cin >> x\_scale >> y\_scale >> x\_fixed >> y\_fixed;

            drawAxis();

            drawPolygon();

            drawScaledFixedPointPolygon(x\_scale, y\_scale, x\_fixed, y\_fixed);

        }

        if (choice == 6)

        {

            drawAxis();

            drawPolygon();

            drawReflectionXPolygon();

        }

        if (choice == 7)

        {

            drawAxis();

            drawPolygon();

            drawReflectionYPolygon();

        }

        if (choice == 8)

        {

            drawAxis();

            drawPolygon();

            drawReflectionOriginPolygon();

        }

        if (choice == 9)

        {

            drawAxis();

            drawPolygon();

            drawReflectionXequalsYPolygon();

        }

        if (choice == 10)

        {

            drawAxis();

            cout << "Enter the x-shearing factor: " << endl;

            cin >> x\_shear;

            drawPolygon();

            drawXShearedPolygon(x\_shear);

        }

        if (choice == 11)

        {

            drawAxis();

            cout << "Enter the y-shearing factor: " << endl;

            cin >> y\_shear;

            drawPolygon();

            drawYShearedPolygon(y\_shear);

        }

    glFlush();

}

int main(int argc, char \*\*argv)

{

    cout << "Enter the number of edges: ";

    cin >> edges;

    if (edges == 2)

    {

        polygon = 1;

    }

    else

    {

        polygon = -1;

    }

    cout << "Enter vertices: \n";

    for (int i = 0; i < edges; i++)

    {

        cout << "Enter co-ordinates for vertex " << i + 1 << " (X, Y): ";

        cin >> x >> y;

        x\_coordinates.push\_back(x);

        y\_coordinates.push\_back(y);

    }

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

    glutInitWindowSize(600, 600);

    glutInitWindowPosition(0, 0);

    glutCreateWindow("Basic 2D Transformations");

    myInit();

    glutDisplayFunc(myDisplay);

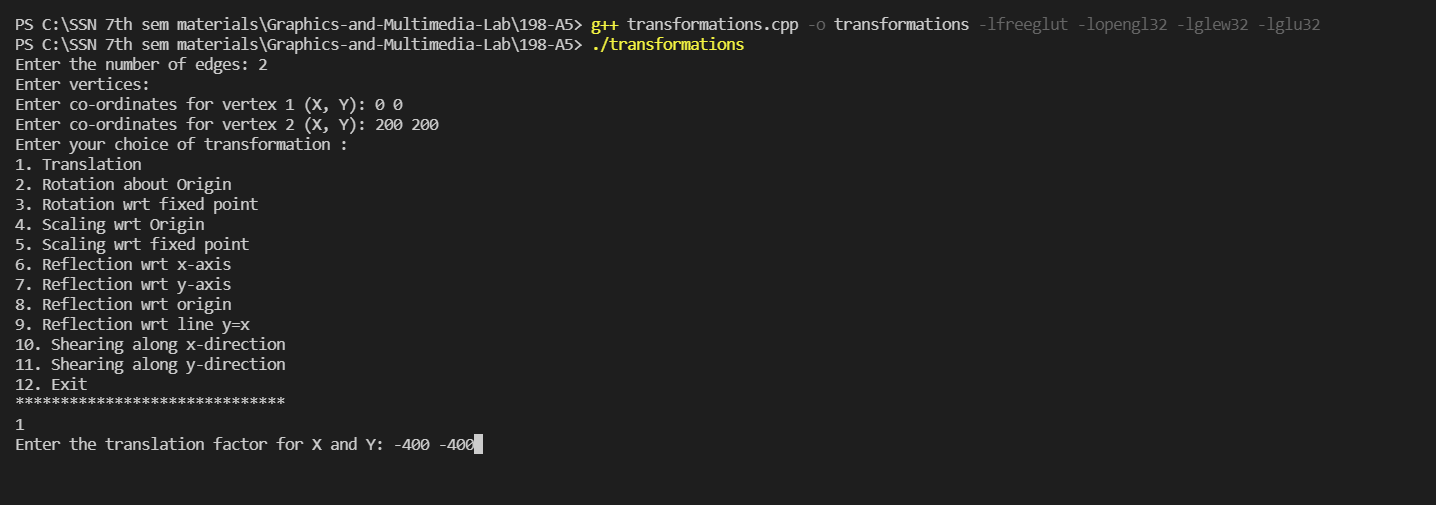
    glutMainLoop();

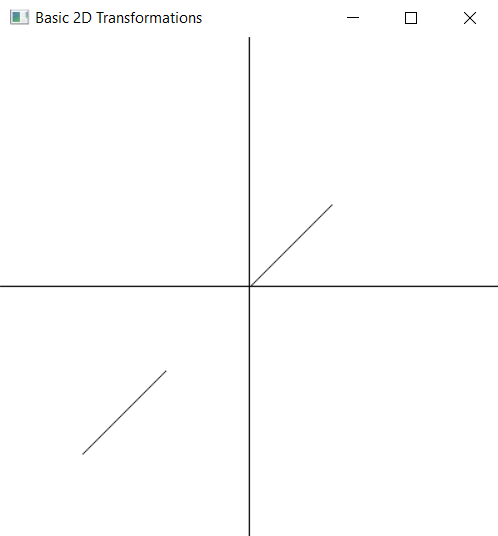
    cout << endl << endl << "Hello";

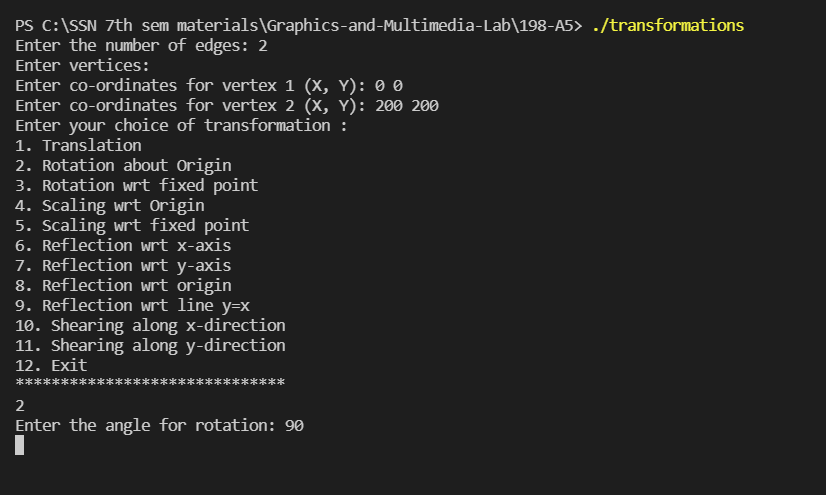
    return 1;

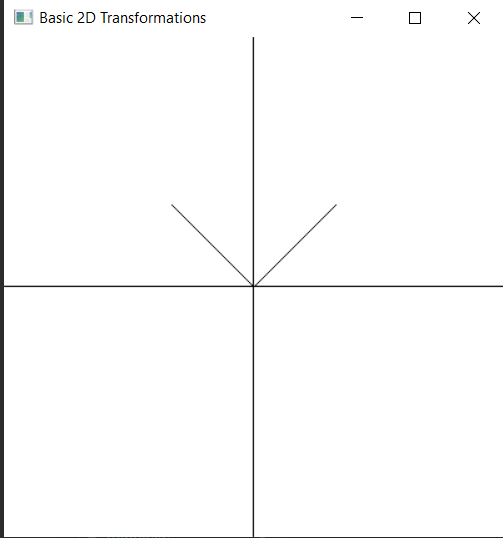
}

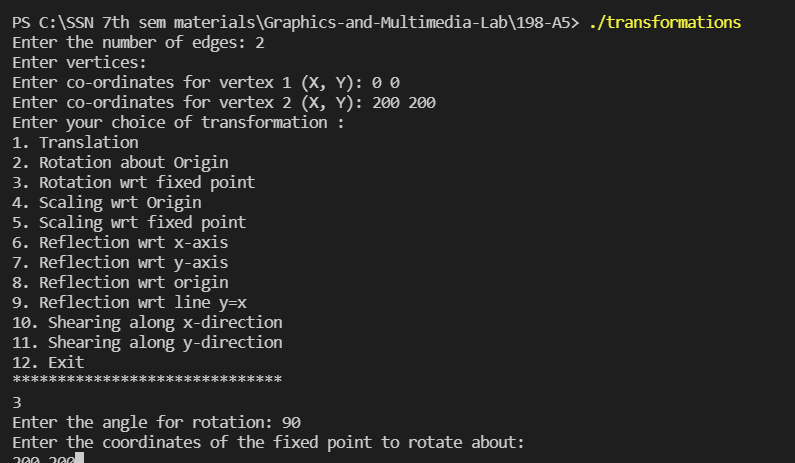
**OUTPUTS:**

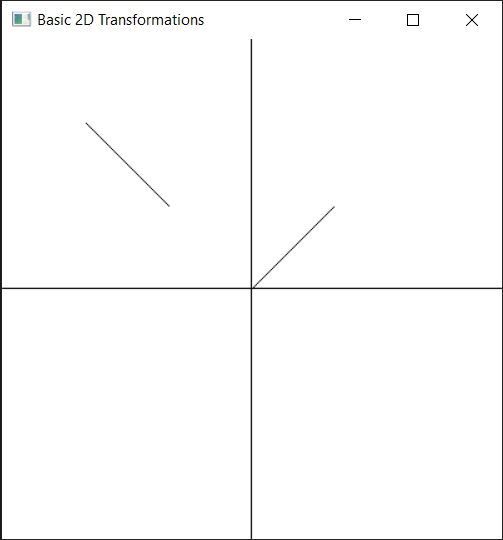
**1. **

****

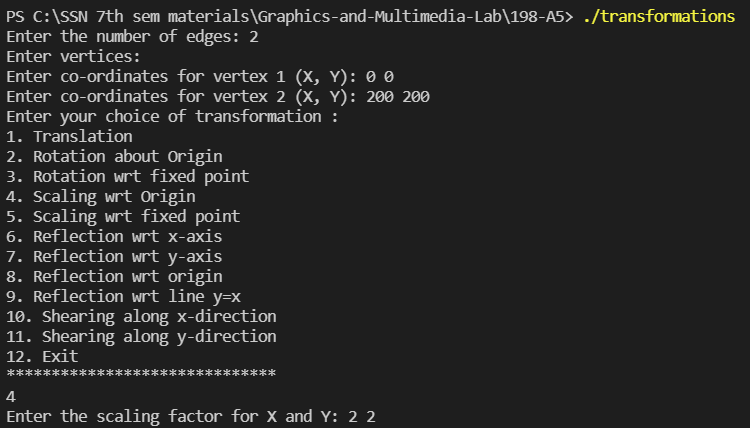
**2. **

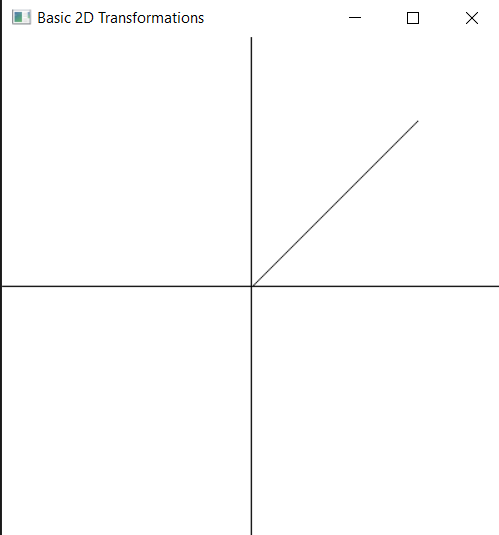
****

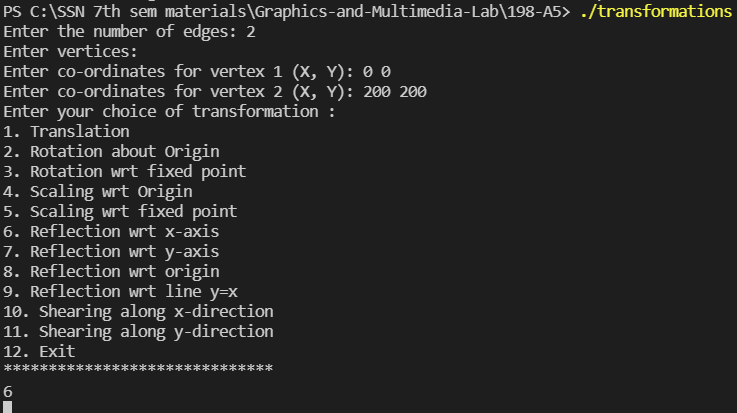
**3. **

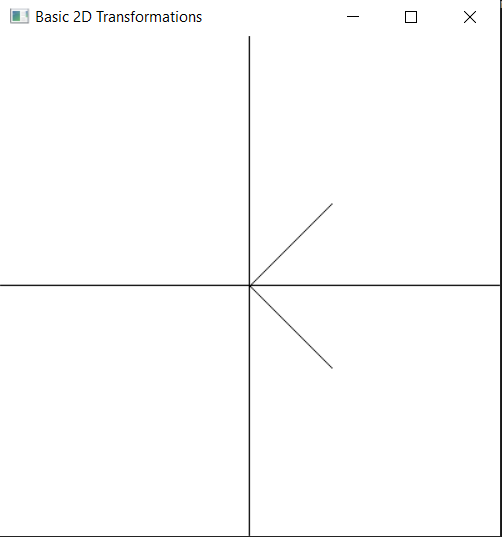
****

**4.**

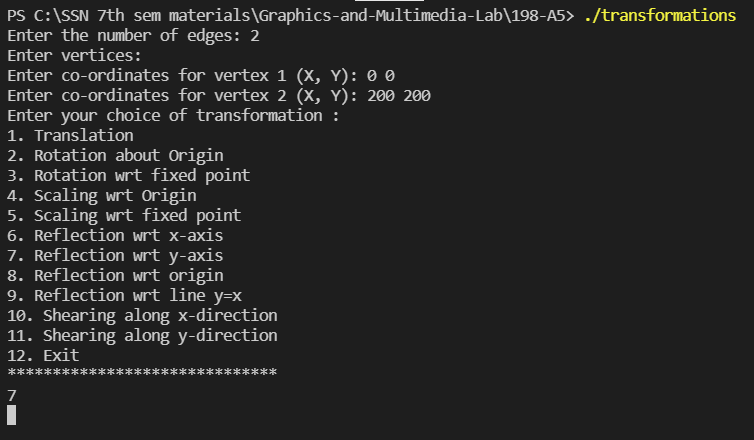
****

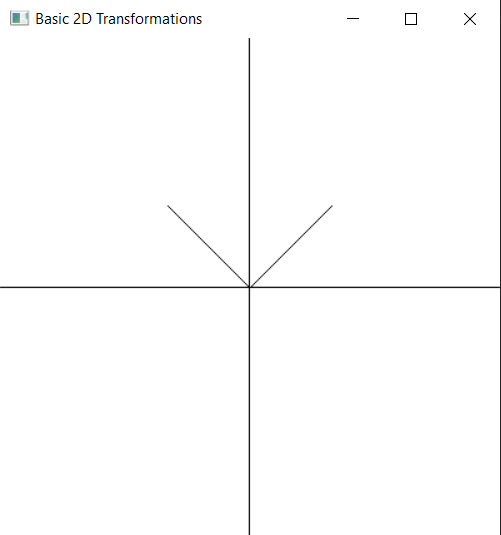
****

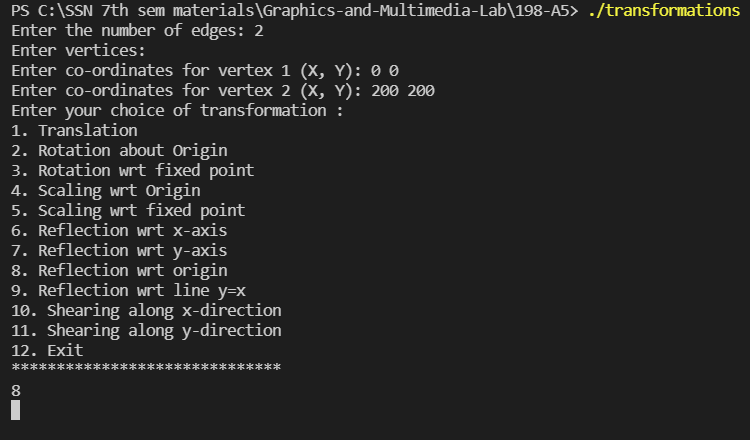
**5. **

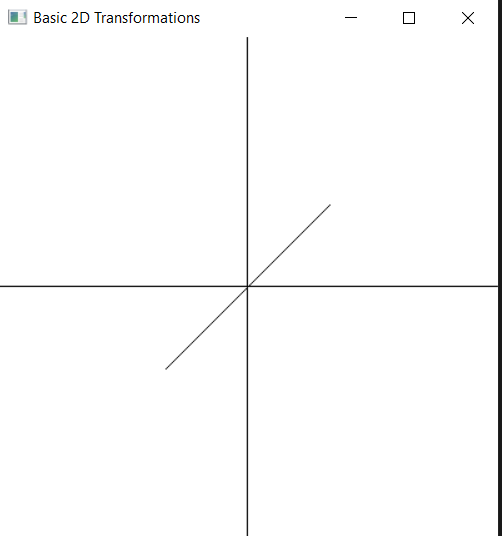
****

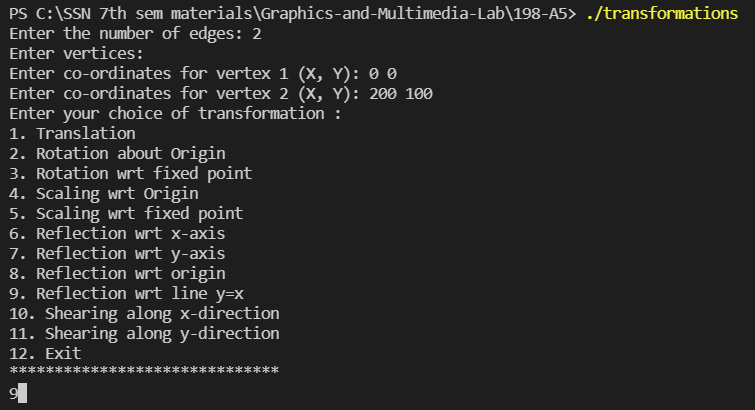
**6.**

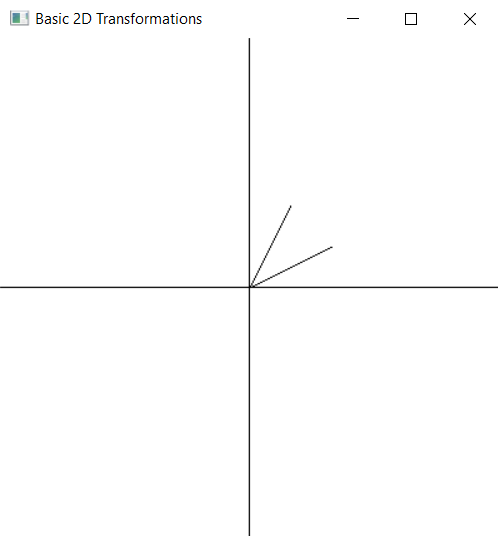
****

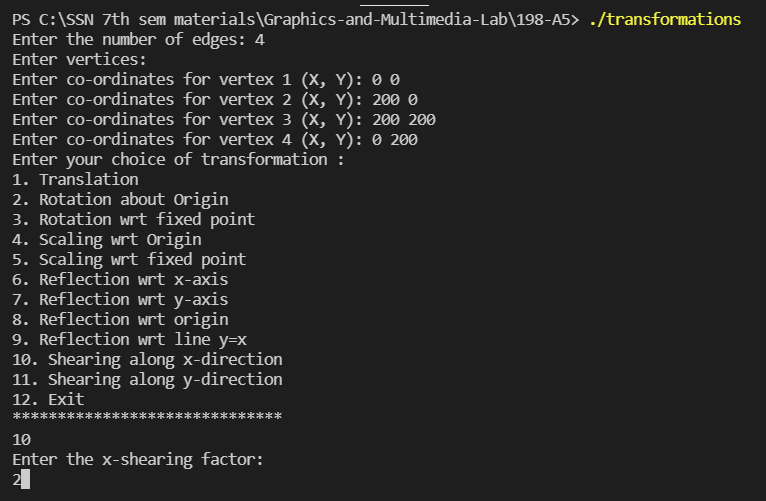
****

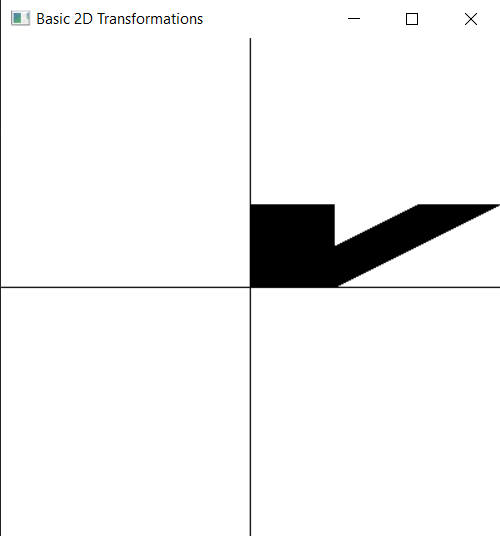
**7. **

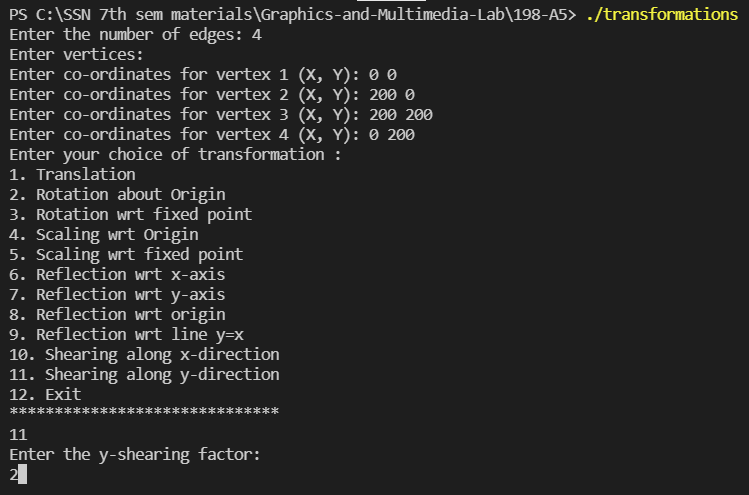
****

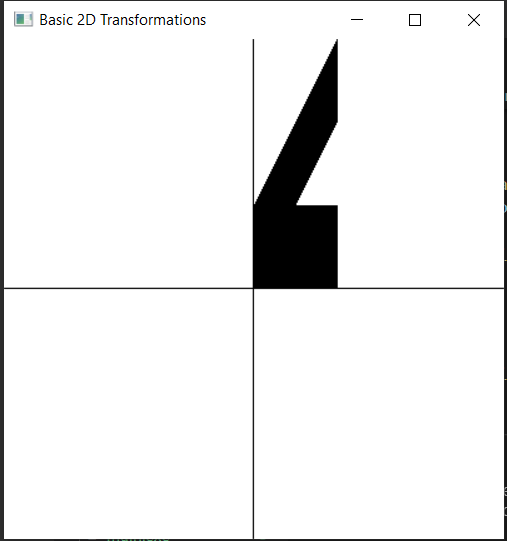
**8. **

****

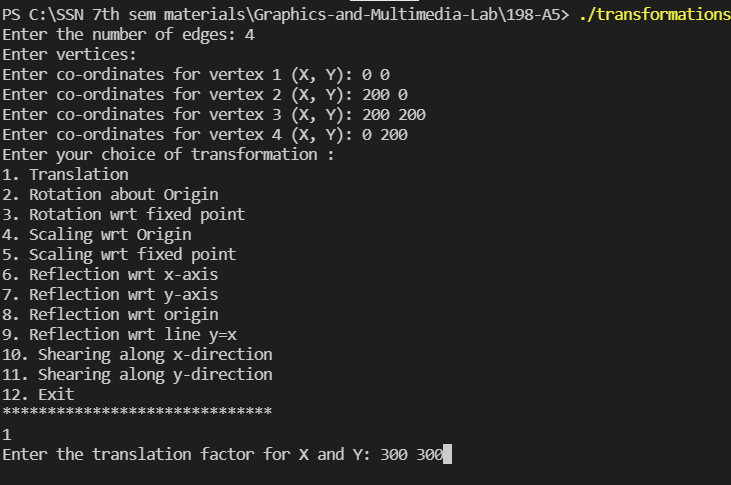
**9. **

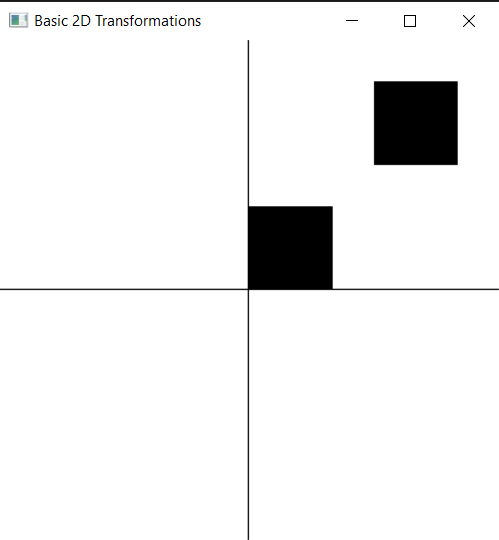
****

**10. **

****

**11.**

****

****