Name	**:**	**V Vikram**	**Class**	**:**	CSE ‘C’
Reg. No.	**:**	**18 5001 194**	**Date**	**:**	06/10/2021
Subject	**:**	**UCS1712---Graphics and Multimedia Lab**			

QUESTION :

Lab Exercise 6:
2D Composite Transformations and Windowing in C++ using OpenGL

a) To compute the composite transformation matrix for any 2 transformations given as input by the user and apply it on the object. The transformation can be any combination of the following.

1) Translation
2) Rotation
3) Scaling
4) Reflection
5) Shearing Display the original and the transformed object.

Calculate the final transformation matrix by multiplying the two individual transformation matrices and then apply it to the object.
Note: Use Homogeneous coordinate representations and matrix multiplication to perform transformations. Divide the output window into four quadrants. (Use LINES primitive to draw x and y axis)

b) Create a window with any 2D object and a different sized viewport. Apply window to viewport transformation on the object. Display both window and viewport.

CODE :-

(a)
Source.cpp :

```
#include<iostream>
#include<GL/glut.h>
#include<vector>
#include "Source.h"
using namespace std;
constexpr auto PI = 3.14;
int n;

vector<pair<int, int>> coords;
int tx, ty;
int xr, yr;
int xf, yf;
double sx, sy;
double ang, angRad;
double shx, shy, sh;
int opt_1, opt_2;
char rfl, shd;
vector<vector<double>> Transformation(3, vector<double>(3, 0)); //final
TRANSFORMATION matrix
vector<vector<double>> trans_1(3, vector<double>(3, 0));
vector<vector<double>> trans_2(3, vector<double>(3, 0));
//Matrices for the 2 transformations

#include"Header.h"

void myInit(void) {
	glClearColor(1.0, 1.0, 1.0, 1.0);
	glColor3f(0.0f, 0.0f, 0.0f);
	glPointSize(4.0);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(-700, 700, -700, 700);
}

void myDisplay() {
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(0.0, 0.0, 0.0);
	draw_X_Y_plane();
	drawPolygon();
	trans_1 = executeTransformMatrix(opt_1);
	trans_2 = executeTransformMatrix(opt_2);
	draw_Transformed_Polygon(trans_1,trans_2);
	glFlush();
```

```
}
void menu_driven(int opr) {

	switch (opr) {
		case 1: {
			cout << "\n\n\t\tTranslation factor : ";
			cin >> tx >> ty;
			break;
		}
		case 2: {
			cout << "\n\n\t\tAngle of rotation : ";
			cin >> ang;
			angRad = ang * PI / 180;
			cout << "\n\t\tRotate about : ";
			cin >> xr >> yr;
			break;
		}
		case 3: {
			cout << "\n\n\t\tScaling factor : ";
			cin >> sx >> sy;
			cout << "\n\t\tScale about : ";
			cin >> xf >> yf;
			break;
		}
		case 4: {
			cout << "\n\t\tReflect about : " ;
			cout << "\n\t\t\ta) X-axis" ;
			cout << "\n\t\t\tb) Y-axis" ;
			cout << "\n\t\t\tc) origin" ;
			cout << "\n\t\t\td) X=Y line  -> " ;
			cin >> rfl;
			break;
		}
		case 5: {
			cout << "\n\t\tShear about : " ;
			cout << "\n\t\t\ta) X-direction" ;
			cout << "\n\t\t\tb) Y-direction -> " ;
			cin >> shd;
			cout << "\n\t\t\tEnter shear parameter : ";
			cin >> sh;
			break;
		}
		default: cout << "\n\n\t\tINVALID INPUT !!!";
	}
}
int main(int argc, char** argv) {
	cout << "\n\t-------------------------------------------";
	cout << "\n\tEx-6 Composite Transformations and Windowing";
	cout << "\n\t-------------------------------------------";
	cout << "\n\nNo. of vertices : ";
	cin >> n;
	int x, y;
	for (int i = 0; i < n; i++) {
```

```
            cout << "\n\tVertex " << i + 1 << " : ";
            cin >> x >> y;
            coords.push_back(make_pair(x, y));
      }
      cout << "\n\nOptions :-";
      cout << "\n\t1) Translation";
      cout << "\n\t2) Rotation";
      cout << "\n\t3) Scaling with respect to";
      cout << "\n\t4) Reflection with respect to";
      cout << "\n\t5) Shearing";
      cout << "\n\tSelect option -> ";

      cout << "\n\n\tOption 1 : ";
      cin >> opt_1;
      menu_driven(opt_1);

      cout << "\n\tOption 2 : ";
      cin >> opt_2;
      menu_driven(opt_2);

      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
      glutInitWindowSize(700, 700);
      glutCreateWindow("Ex-6 Composite Transformations and Windowing");
      glutDisplayFunc(myDisplay);
      myInit();
      glutMainLoop();
      return 0;
}
```

Header.h:

```
#pragma once

void draw_X_Y_plane() {

      glBegin(GL_LINES);
      glColor3f(0.0, 0.0, 0.0);
      glVertex2d(-700, 0);
      glVertex2d(700, 0);
      glVertex2d(0, -700);
      glVertex2d(0, 700);
      glEnd();
}
void drawPolygon()
{
```

```
	glBegin(GL_LINE_LOOP);
	glColor3f(0.0/255.0f, 0.0 / 255.0f, 102.0 / 255.0f);
	for (int i = 0; i < n; i++)
	{
		glVertex2d(coords[i].first, coords[i].second);
	}
	glEnd();
}
vector<vector<double>> translate()
{
	vector<vector<double>> temp(3, vector<double>(3, 0));

	temp[0][0] = 1;
	temp[1][1] = 1;
	temp[2][2] = 1;
	temp[0][2] = tx;
	temp[1][2] = ty;

	return temp;
}
vector<vector<double>> rotate_fixed() {
	vector<vector<double>> temp(3, vector<double>(3, 0));

	temp[0][0] = cos(angRad);
	temp[1][1] = cos(angRad);
	temp[2][2] = 1;
	temp[1][0] = sin(angRad);
	temp[0][1] = -1 * sin(angRad);
	temp[0][2] = xr * (1 - cos(angRad)) + yr * sin(angRad);
	temp[1][2] = yr * (1 - cos(angRad)) - xr * sin(angRad);

	return temp;
}
vector<vector<double>>  scale_fixed() {
	vector<vector<double>> temp(3, vector<double>(3, 0));

	temp[0][0] = sx;
	temp[1][1] = sy;
	temp[2][2] = 1;
	temp[0][2] = xf * (1 - sx);
	temp[1][2] = yf * (1 - sy);

	return temp;
}
vector<vector<double>> reflect_X() {
	vector<vector<double>> temp(3, vector<double>(3, 0));

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

	return temp;
}
```

```
vector<vector<double>> reflect_Y() {
	vector<vector<double>> temp(3, vector<double>(3, 0));

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

	return temp;
}
vector<vector<double>> reflect_Origin() {
	vector<vector<double>> temp(3, vector<double>(3, 0));

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

	return temp;
}
vector<vector<double>> reflect_X_Y() {
	vector<vector<double>> temp(3, vector<double>(3, 0));

	temp[0][1] = 1;
	temp[1][0] = 1;
	temp[2][2] = 1;

	return temp;
}
vector<vector<double>> shear_X() {
	vector<vector<double>> temp(3, vector<double>(3, 0));

	temp[0][0] = 1;
	temp[1][1] = 1;
	temp[2][2] = 1;
	temp[0][1] = sh;

	return temp;
}
vector<vector<double>> shear_Y() {
	vector<vector<double>> temp(3, vector<double>(3, 0));

	temp[0][0] = 1;
	temp[1][1] = 1;
	temp[2][2] = 1;
	temp[1][0] = sh;

	return temp;
}
vector<vector<double>> executeTransformMatrix(int opt) {
	vector<vector<double>> trans_mat(3, vector<double>(3, 0));

	if (opt == 1)		trans_mat = translate();
	else if (opt == 2)	trans_mat = rotate_fixed();
	else if (opt == 3)	trans_mat = scale_fixed();
```

```
	else if (opt == 4) {
		if (rfl == 'a')		trans_mat = reflect_X();
		else if (rfl == 'b') trans_mat = reflect_Y();
		else if (rfl == 'c') trans_mat = reflect_Origin();
		else if (rfl == 'd') trans_mat = reflect_X_Y();
	}
	else {
		if (shd == 'a')		trans_mat = shear_X();
		else			trans_mat = shear_Y();
	}

	return trans_mat;
}
void draw_Transformed_Polygon(vector<vector<double>> trans_1,
vector<vector<double>> trans_2) {

	/*Multiply the two transformation matrices to find the
	final TRANSFORMATION matrix*/
	for (int i = 0; i < 3; i++) {
		for (int j = 0; j < 3; j++) {
			for (int k = 0; k < 3; k++) {
				Transformation[i][j] += trans_1[i][k] *
trans_2[k][j];
			}
		}
	}

	glBegin(GL_LINE_LOOP);
	glColor3f(1.0, 0.0, 0.0);
	vector<pair<int, int>> newVertices;
	vector<double> curpoint(3, 0), matProduct(3, 0);
	for (int i = 0; i < n; i++) {
		curpoint[0] = coords[i].first;
		curpoint[1] = coords[i].second;
		curpoint[2] = 1;
		matProduct[0] = 0;
		matProduct[1] = 0;
		matProduct[2] = 0;
		for (int j = 0; j < 3; j++) {
			for (int k = 0; k < 3; k++) {
				//Apply the TRANSFORMATION matrix to all the
vertices
				matProduct[j] += Transformation[j][k] *
curpoint[k];
			}
		}
		newVertices.push_back(make_pair(round(matProduct[0]),
round(matProduct[1])));

	}
	for (int i = 0; i < n; i++)
	{
		glVertex2d(newVertices[i].first, newVertices[i].second);
```

```
	}
	glEnd();
}
```

(b)
window_to_view.cpp :

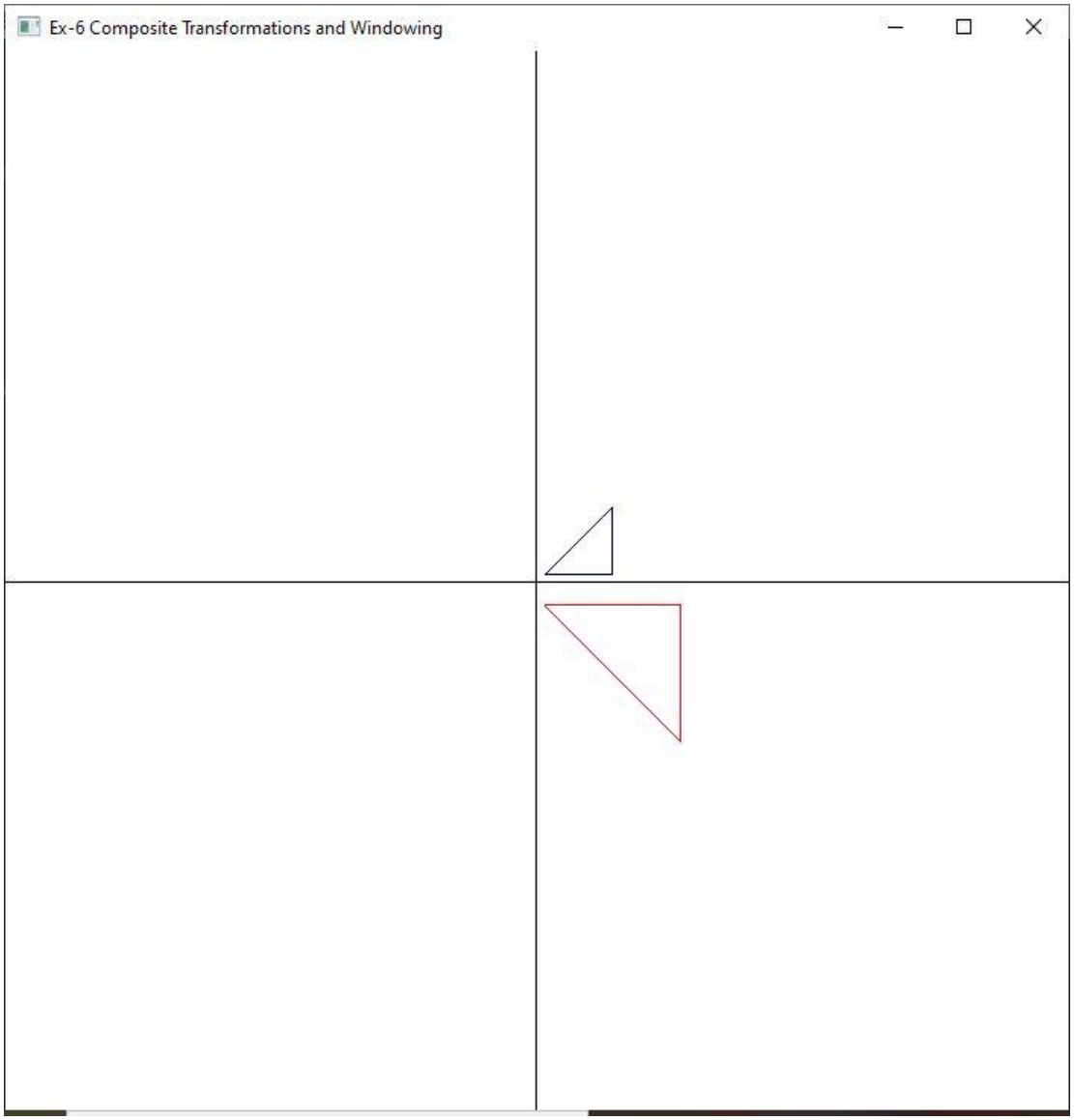
```
#include<vector>
#include<iostream>
#include<GL/glut.h>
#include<utility>
#include<math.h>
using namespace std;
int n;
vector<pair<int, int>> coordinates;

int xw_min = 0, yw_min = 0, xw_max = 1000, yw_max = 1000;
int xv_min, xv_max, yv_min, yv_max;
double sx, sy;

void myInit(void) {
	glClearColor(1.0, 1.0, 1.0, 1.0);
	glColor3f(1.0f, 0.0f, 0.0f);
	glPointSize(4.0);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0, 720,0, 720);
}
void draw_ViewPort_Window() {
	glBegin(GL_LINE_LOOP);
	glColor3f(0.0f, 0.0f, 0.0f);
	glVertex2d(xv_min, yv_min);
	glVertex2d(xv_min, yv_max);
	glVertex2d(xv_max, yv_max);
	glVertex2d(xv_max, yv_min);
	glEnd();
}
void draw_Window_Object() {
	glBegin(GL_LINE_LOOP);
	glColor3f(1.0f, 0.0f, 0.0f);
	for (int i = 0; i < n; i++) {
		glVertex2d(coordinates[i].first, coordinates[i].second);
	}
	glEnd();
}
void draw_ViewPort_Object() {
```

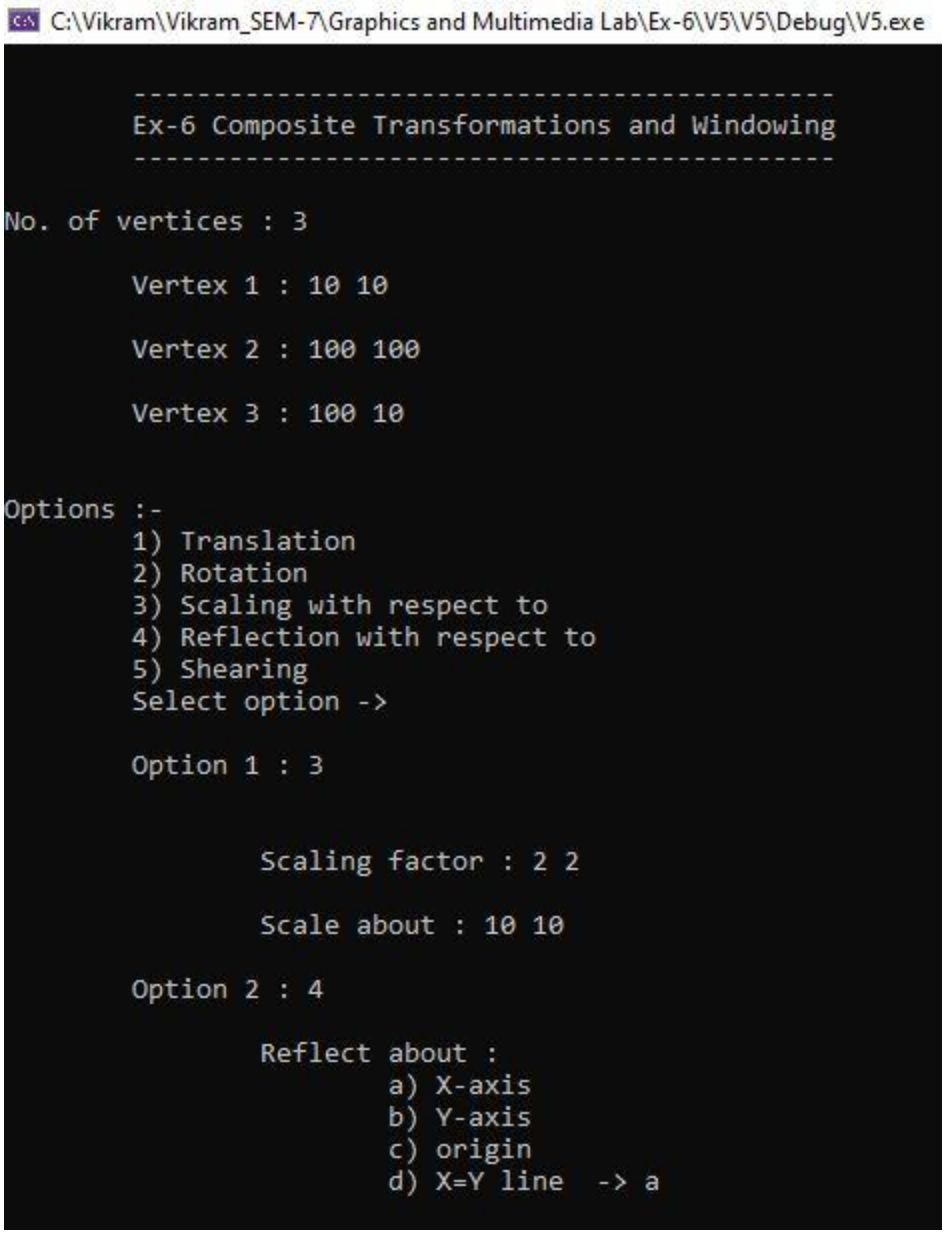
```
	glBegin(GL_LINE_LOOP);
	glColor3f(1.0f, 0.0f, 1.0f);
	for (int i = 0; i < n; i++) {
		glVertex2d(xv_min + (coordinates[i].first - xw_min) * sx,
yv_min + (coordinates[i].second - yw_min) * sy);
	}
	glEnd();
}
void myDisplay() {
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(0.0, 0.0, 0.0);
	draw_Window_Object();
	draw_ViewPort_Object();
	draw_ViewPort_Window();
	glFlush();
}
int main(int argc, char** argv) {

	cout << "\n\t\t--------------------------------------";
	cout << "\n\t\tEx:6 (b) Windowing in C++ using OpenGL" ;
	cout << "\n\t\t--------------------------------------";

	cout << "\n\n\tNo. of vertices of OBJECT : ";
	cin >> n;
	int x, y;
	for (int i = 0; i < n; i++) {
		cout << "\n\tVertex_" << i + 1 << " : ";
		cin >> x >> y;
		coordinates.push_back(make_pair(x, y));
	}
	cout << "\n\tEnter Viewport details";
	cout << "\n\n\t\tMin x & Max x : ";
	cin >> xv_min >> xv_max;
	cout << "\n\t\tMin y & Max y : ";
	cin >> yv_min >> yv_max;

	sx = (xv_max - xv_min) * 1.0 / (xw_max - xw_min);
	sy = (yv_max - yv_min) * 1.0 / (yw_max - yw_min);

	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize(720, 720);
	glutCreateWindow("Window to Viewport Transformation");
	glutDisplayFunc(myDisplay);
	myInit();
	glutMainLoop();
	return 0;
}
```

OUTPUT SNAPSHOTS :

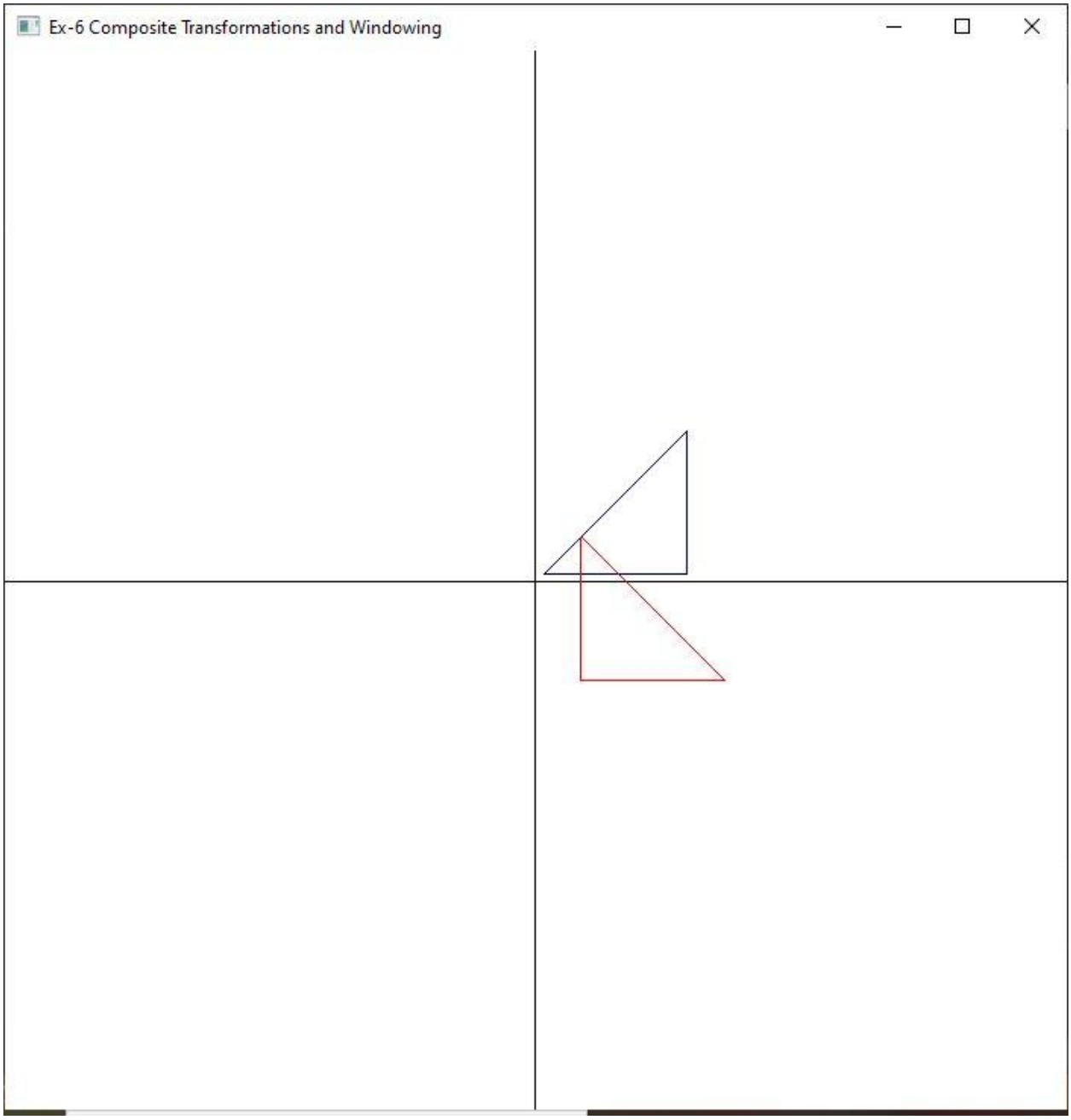
(A) Composite transformations

1.Scaling -> Reflection :




C:\Vikram\Vikram_SEM-7\Graphics and Multimedia Lab\Ex-6\V5\V5\Debug\V5.exe
Ex-6 Composite Transformations and Windowing
No. of vertices : 3
Vertex 1 : 10 10
Vertex 2 : 100 100
Vertex 3 : 100 10
Options :-
1) Translation
2) Rotation
3) Scaling with respect to
4) Reflection with respect to
5) Shearing
Select option ->
Option 1 : 3
Scaling factor : 2 2
Scale about : 10 10
Option 2 : 4
Reflect about :
a) X-axis
b) Y-axis
c) origin
d) X=Y line -> a

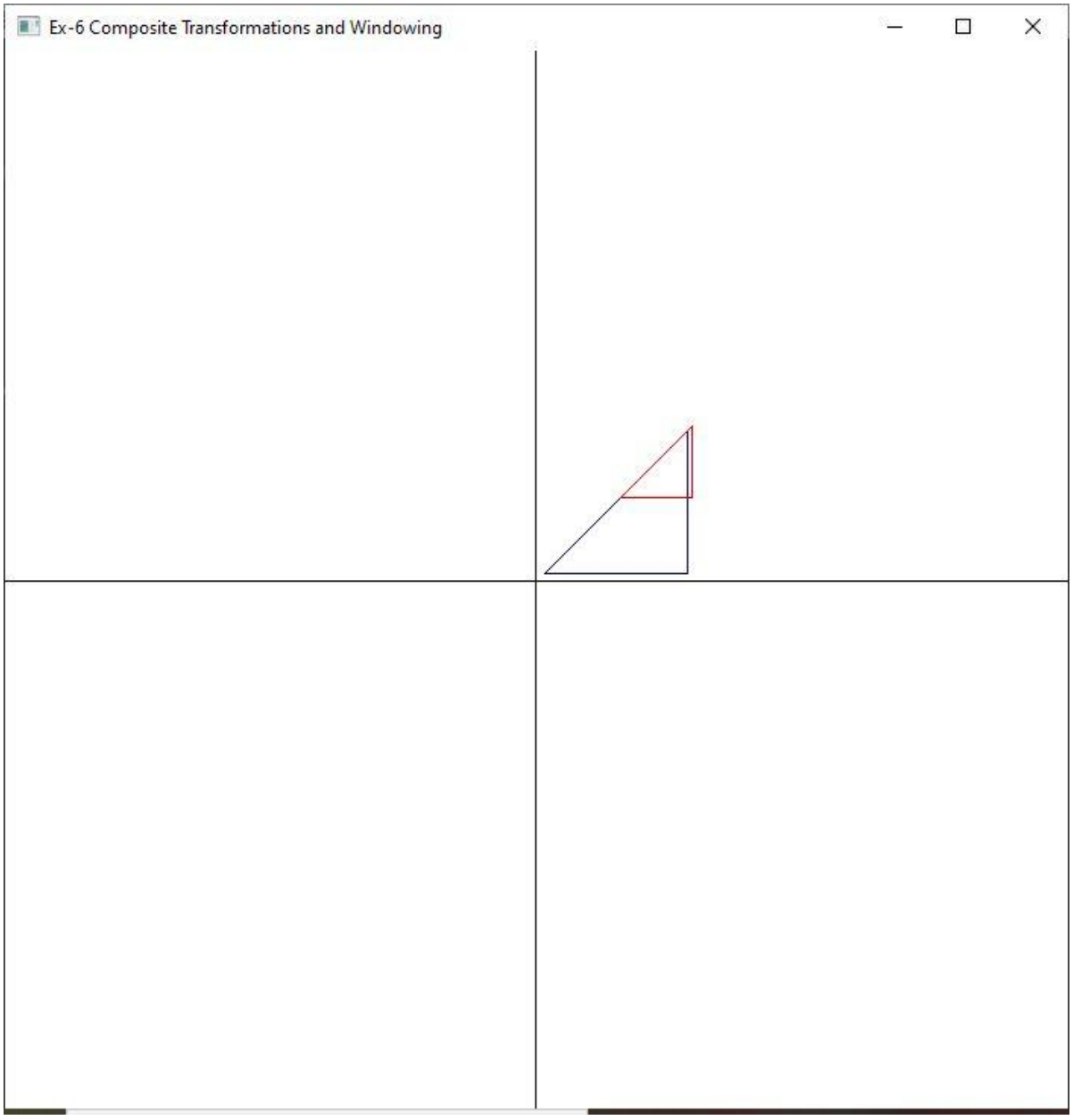
2.Translation -> Rotation :

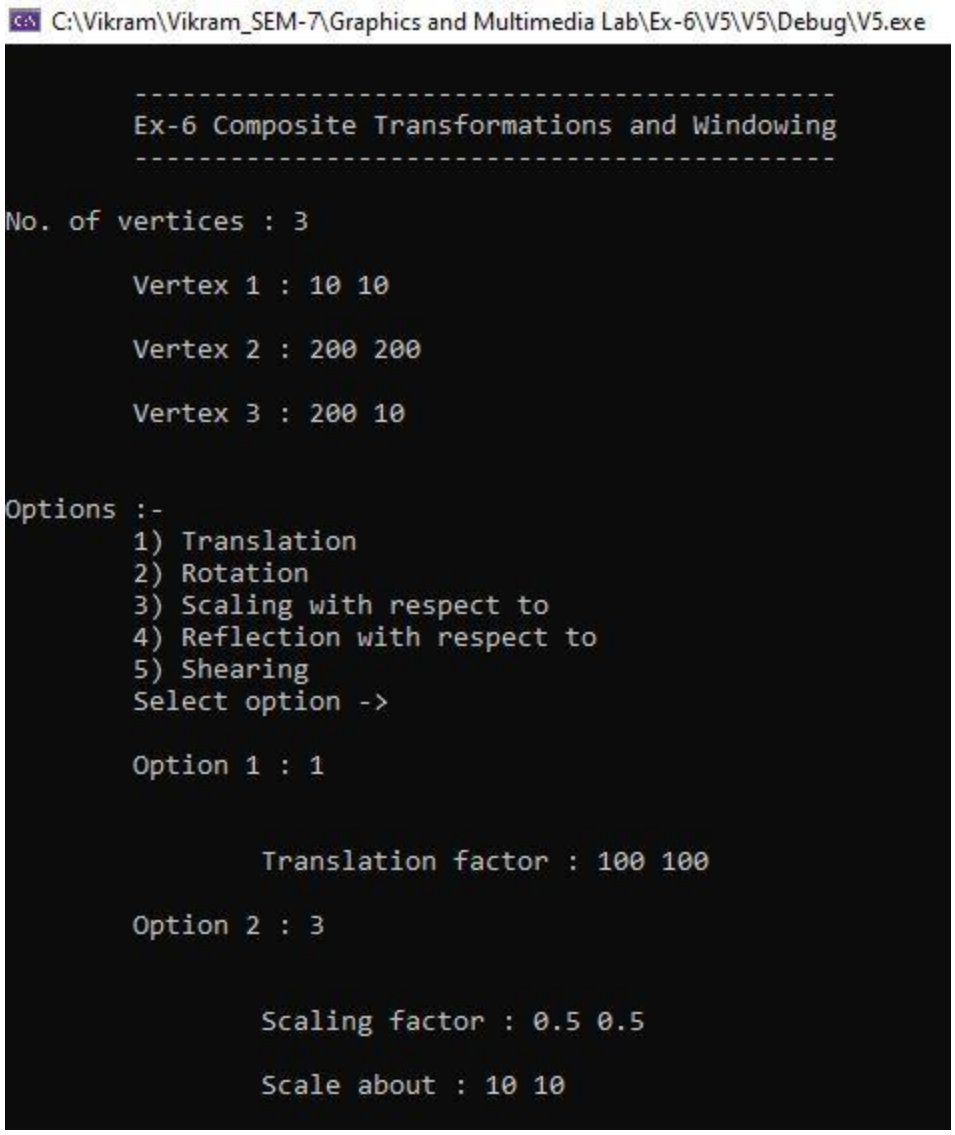


C:\Vikram\Vikram_SEM-7\Graphics and Multimedia Lab\Ex-6\V5\V5\Debug\V5.exe

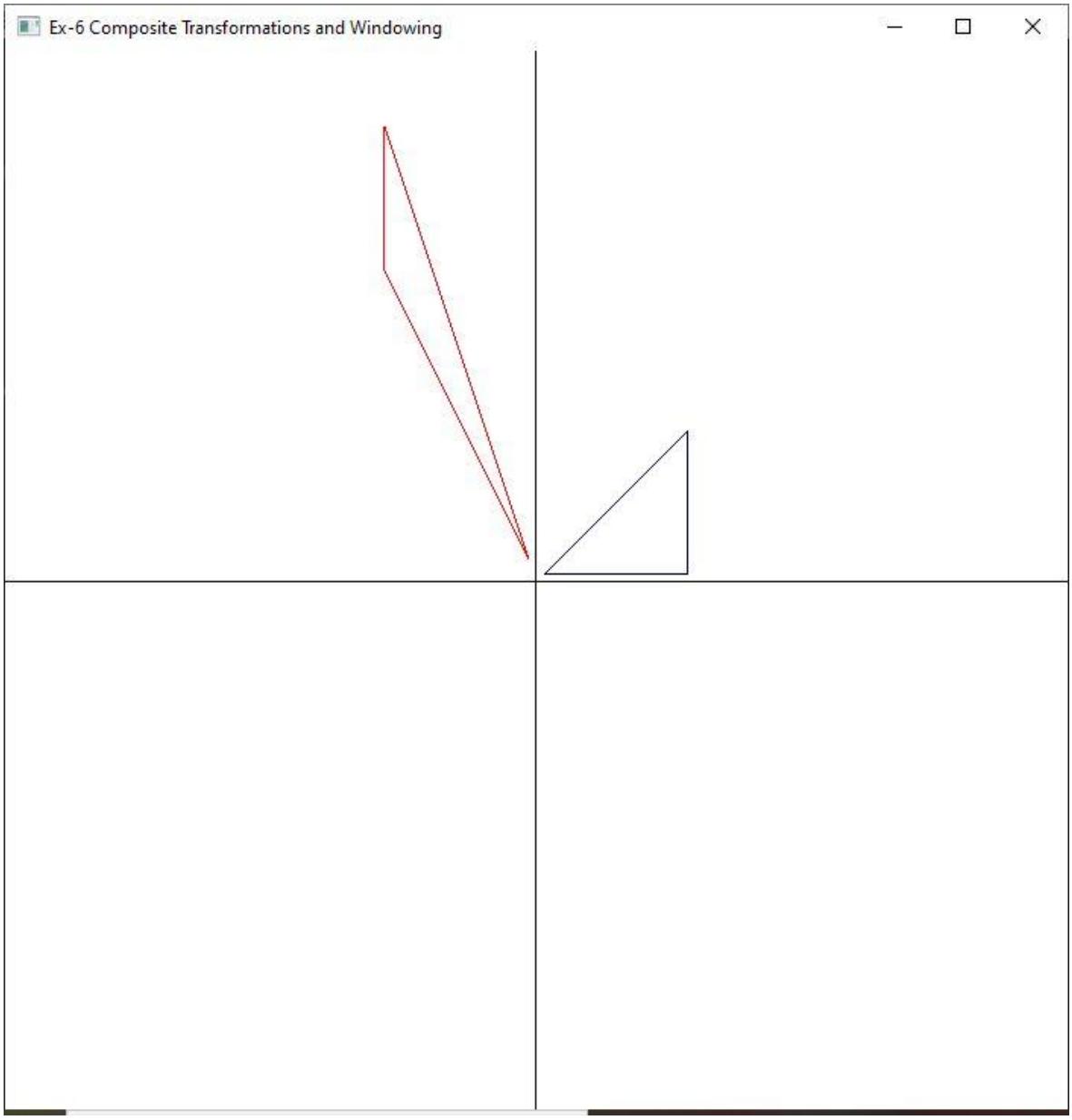
```
        ------------------------------------------
        Ex-6 Composite Transformations and Windowing
        ------------------------------------------

No. of vertices : 3

        Vertex 1 : 10 10

        Vertex 2 : 200 200

        Vertex 3 : 200 10


Options :-
        1) Translation
        2) Rotation
        3) Scaling with respect to
        4) Reflection with respect to
        5) Shearing
        Select option ->

        Option 1 : 1


                Translation factor : 50 50

        Option 2 : 2


                Angle of rotation : -90

                Rotate about : 10 10
```

3.Translation -> Scaling:




C:\Vikram\Vikram_SEM-7\Graphics and Multimedia Lab\Ex-6\V5\V5\Debug\V5.exe
Ex-6 Composite Transformations and Windowing
No. of vertices : 3
Vertex 1 : 10 10
Vertex 2 : 200 200
Vertex 3 : 200 10
Options :-
1) Translation
2) Rotation
3) Scaling with respect to
4) Reflection with respect to
5) Shearing
Select option ->
Option 1 : 1
Translation factor : 100 100
Option 2 : 3
Scaling factor : 0.5 0.5
Scale about : 10 10

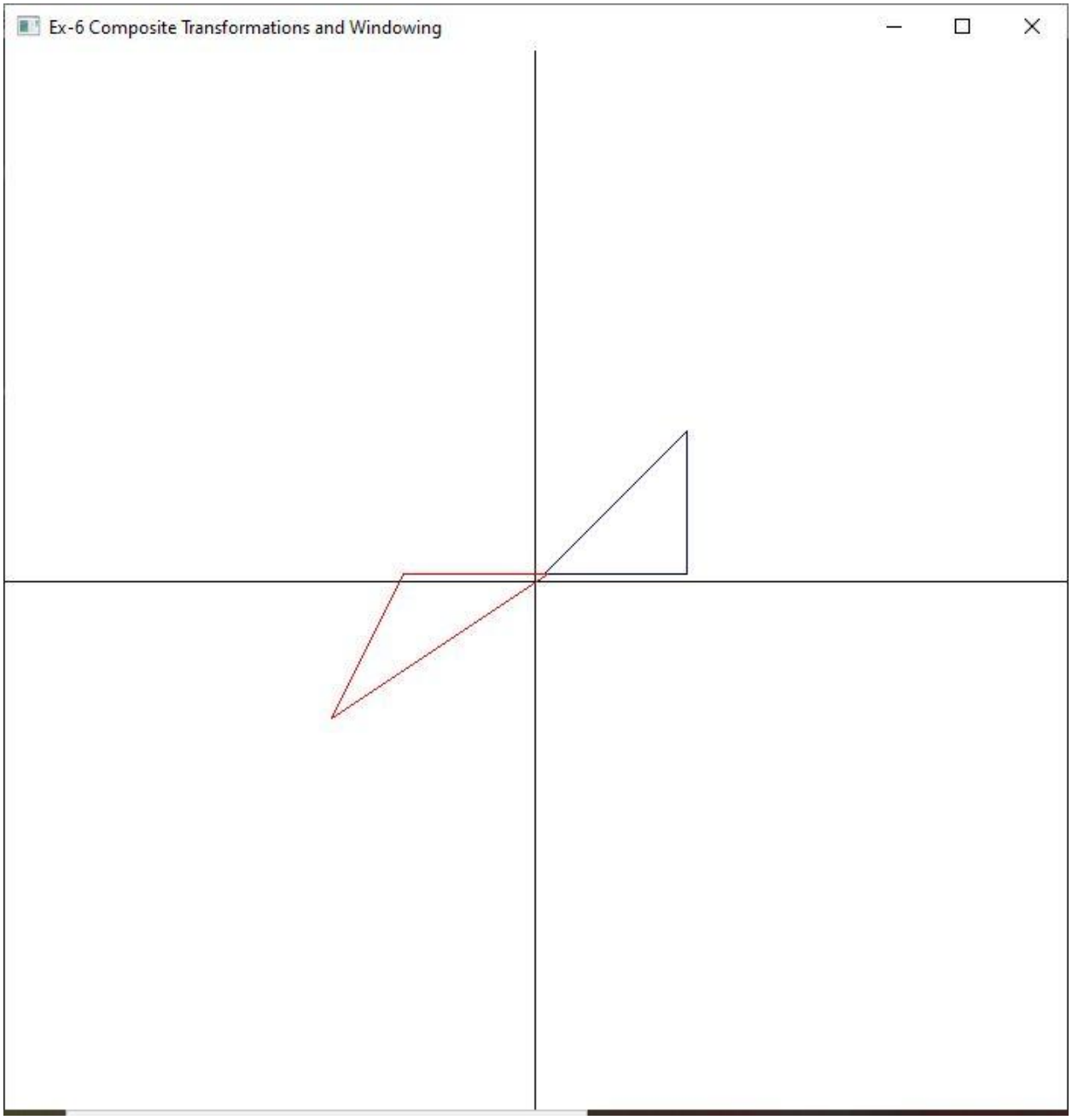
4.Reflection ->Shearing :



C:\Vikram\Vikram_SEM-7\Graphics and Multimedia Lab\Ex-6\V5\V5\Debug\V5.exe

```
        ------------------------------------------
        Ex-6 Composite Transformations and Windowing
        ------------------------------------------

No. of vertices : 3

        Vertex 1 : 10 10

        Vertex 2 : 200 200

        Vertex 3 : 200 10


Options :-
        1) Translation
        2) Rotation
        3) Scaling with respect to
        4) Reflection with respect to
        5) Shearing
        Select option ->

        Option 1 : 4

                Reflect about :
                        a) X-axis
                        b) Y-axis
                        c) origin
                        d) X=Y line  -> b

        Option 2 : 5

                Shear about :
                        a) X-direction
                        b) Y-direction -> b

                        Enter shear parameter : 2 2
```

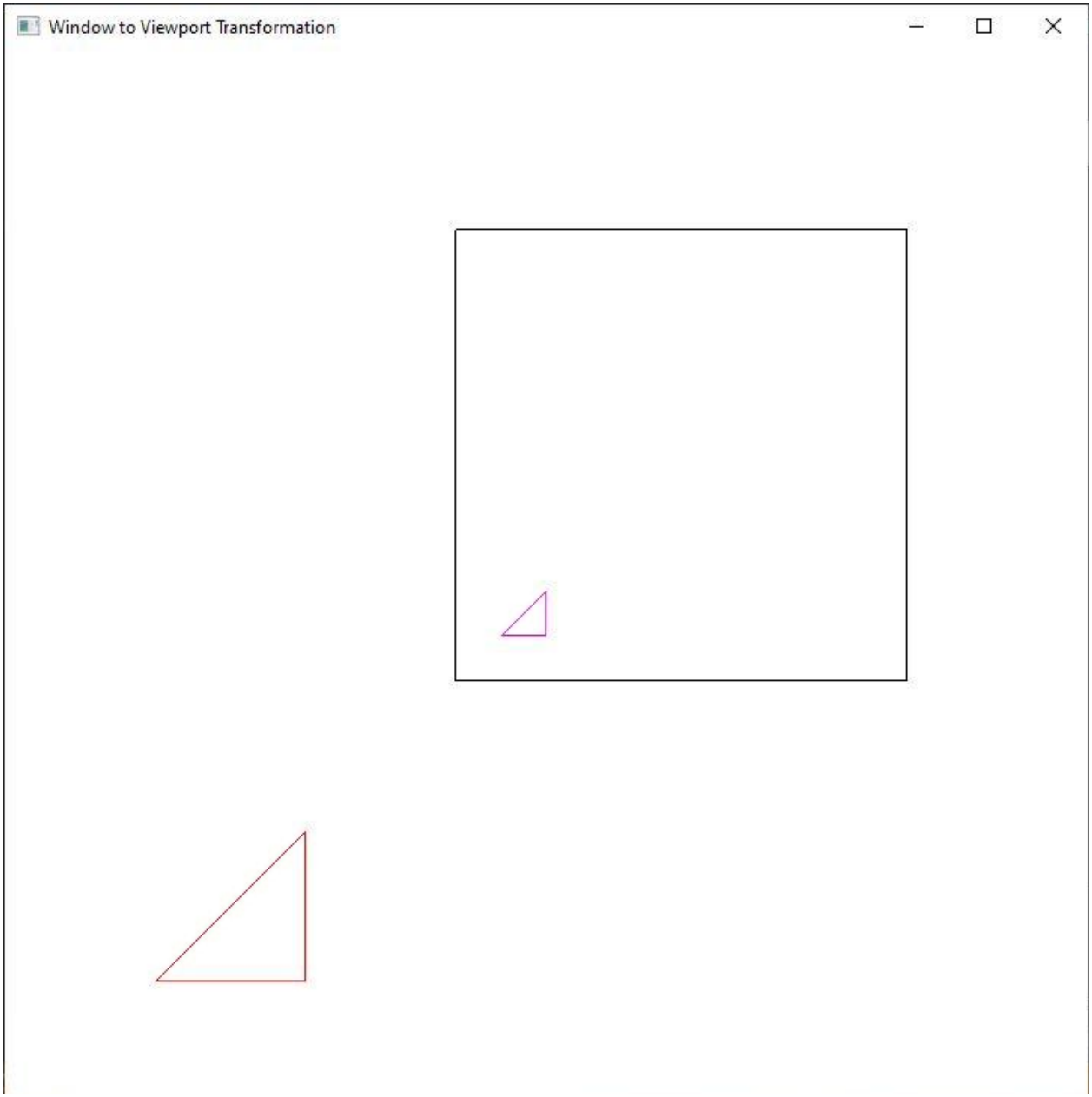
5.Shearing -> Rotation :

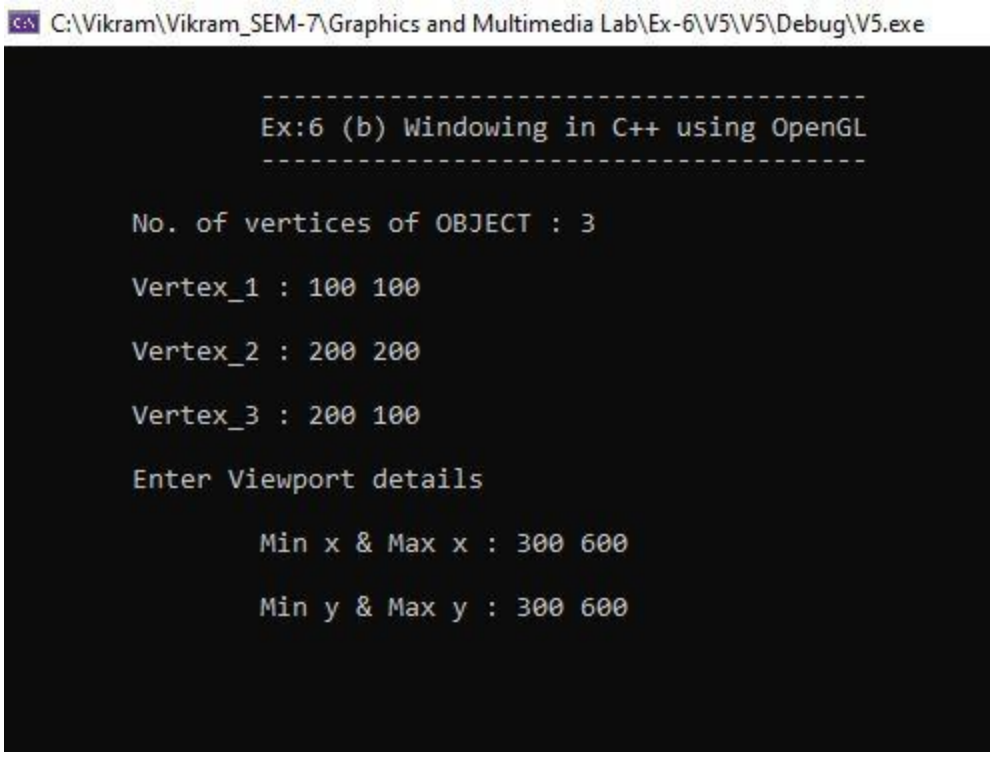


C:\Vikram\Vikram_SEM-7\Graphics and Multimedia Lab\Ex-6\V5\V5\Debug\V5.exe

```
        ------------------------------------------
        Ex-6 Composite Transformations and Windowing
        ------------------------------------------

No. of vertices : 3

        Vertex 1 : 10 10

        Vertex 2 : 200 200

        Vertex 3 : 200 10


Options :-
        1) Translation
        2) Rotation
        3) Scaling with respect to
        4) Reflection with respect to
        5) Shearing
        Select option ->

        Option 1 : 5

                Shear about :
                        a) X-direction
                        b) Y-direction -> a

                        Enter shear parameter : 0.5

        Option 2 : 2


                Angle of rotation : 180

                Rotate about : 10 10
```

(B) Window to Viewport Transformation :





CONCLUSION :

Thus

a) The composite transformation matrix for any 2 transformations given as input by the user was computed and applied on the object, successfully.

b) A window with any 2D object and a different sized viewport was created and, window to viewport transformation on the object was applied successfully , and both window and viewport were displayed.