Name	**:**	**V Vikram**	**Class**	**:**	CSE ‘C’
Reg. No.	**:**	**18 5001 194**	**Date**	**:**	13/09/2021
Subject	**:**	**UCS1712---Graphics and Multimedia Lab**			

QUESTION :

Lab Exercise 5

2D Transformations in C++ using OpenGL

To apply the following 2D transformations on objects and to render the final output along with the original object.

1) Translation
2) Rotation
    a) about origin
    b) with respect to a fixed point (xr,yr)
3) Scaling with respect to
    a) origin - Uniform Vs Differential Scaling
    b) fixed point (xf,yf)
4) Reflection with respect to
    a) x-axis
    b) y-axis
    c) origin
    d) the line x=y
5) Shearing
    a) x-direction shear
    b) y-direction shear

CODE :-

Main.cpp :

```
#include<vector>
#include<GL/glut.h>
#include<iostream>
using namespace std;

const int pi = 3.14;
int n,opt=0;
int tx, ty; //translation factors
int xr, yr; //rotation factors
int xf, yf; //scaling factors
double ang, angRad; //radian angle
double sx, sy;
double shx, shy; //shear factors
vector<pair<int, int>> vertices;
#include"temp_header.h"

void myInit(void) {

    glClearColor(1.0, 1.0, 1.0, 1.0);
    glColor3f(255.0f / 255.0f, 255.0f / 255.0f, 255.0f / 255.0f);
    glPointSize(4.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-500.0, 500.0, -500.0, 500.0);
}



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

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 150);

    glutCreateWindow("Ex5 - 2D Transformations");
    cout << "\n\t------------------------";
    cout << "\n\tEx5 - 2D Transformations";
    cout << "\n\t------------------------";
    cout << "\nNo. of Vertices : ";
    cin >> n;

    int x, y;
    for (int i = 0; i < n; i++) {
        cout << "\nVertex_" << i + 1 << " : ";
        cin >> x >> y;
```

```
        vertices.push_back({ x,y });
    }
    cout << "\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 -> ";

    cin >> opt;
    glutDisplayFunc(menu_driven);
    myInit();
    glutMainLoop();
    return 0;
}
```

Header.cpp :

```
#pragma once

void drawPolygon() {

    //X-Y axes
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_LINES);
    glVertex2d(-500, 0);
    glVertex2d(500, 0);
    glVertex2d(0, -500);
    glVertex2d(0, 500);
    glEnd();

    glBegin(GL_LINE_LOOP);
    glColor3f(100.0f / 255.0f, 200.0f / 255.0f, 100.0f / 255.0f);
    glPointSize(4.0);
    for (int i = 0; i < n; i++) {
        int x = vertices[i].first;
        int y = vertices[i].second;
        glVertex2d(x, y);
    }
    glEnd();

}

void translation() {
    glBegin(GL_LINE_LOOP);
```

```
    glColor3f(200.0f / 255.0f, 100.0f / 255.0f, 100.0f / 255.0f);
    glPointSize(4.0);
    for (int i = 0; i < n; i++) {
        int x = vertices[i].first;
        int y = vertices[i].second;
        glVertex2d(x + tx, y + ty);
    }
    glEnd();
}

void rotation_origin() {
    glBegin(GL_LINE_LOOP);
    glColor3f(200.0f / 255.0f, 100.0f / 255.0f, 100.0f / 255.0f);
    glPointSize(4.0);
    for (int i = 0; i < n; i++) {
        int x = vertices[i].first;
        int y = vertices[i].second;
        glVertex2d(round(x * cos(angRad) - y * sin(angRad)),
            round(x * sin(angRad) + y * cos(angRad)));
    }
    glEnd();
}

void scaling_origin() {
    glBegin(GL_LINE_LOOP);
    glColor3f(200.0f / 255.0f, 100.0f / 255.0f, 100.0f / 255.0f);
    glPointSize(4.0);
    for (int i = 0; i < n; i++) {
        int x = vertices[i].first;
        int y = vertices[i].second;
        glVertex2d(x * sx, y * sy);
    }
    glEnd();
}

void rotation_fixed_pt() {
    vector<pair<int, int>> newvertices;
    vector<vector<double>> rotate(3, vector<double>(3, 0));
    vector<double> curr(3, 0), res(3, 0);
    rotate[0][0] = cos(angRad);
    rotate[0][1] = -1 * sin(angRad);
    rotate[0][2] = xr * (1 - cos(angRad)) + yr * sin(angRad);
    rotate[1][0] = sin(angRad);
    rotate[1][1] = cos(angRad);
    rotate[1][2] = yr * (1 - cos(angRad)) - xr * sin(angRad);
    rotate[2][2] = 1;

    for (int i = 0; i < n; i++) {
        curr[0] = vertices[i].first;
        curr[1] = vertices[i].second;
        curr[2] = 1;
        res = vector<double>(3, 0);
        for (int j = 0; j < 3; j++) {
```

```
            for (int k = 0; k < 3; k++) {
                res[j] += rotate[j][k] * curr[k];
            }
        }
        newvertices.push_back({ round(res[0]),round(res[1]) });
    }

    glBegin(GL_LINE_LOOP);
    glColor3f(200.0f / 255.0f, 100.0f / 255.0f, 100.0f / 255.0f);
    glPointSize(4.0);
    for (int i = 0; i < n; i++) {
        int x = newvertices[i].first;
        int y = newvertices[i].second;
        glVertex2d(x, y);
    }
    glEnd();
}

void scaling_fixed_pt() {
    vector<pair<int, int>> newvertices;
    vector<vector<double>> scale(3, vector<double>(3, 0));
    vector<double> curr(3, 0), res(3, 0);
    scale[0][0] = sx;
    scale[0][2] = xf * (1 - sx);
    scale[1][1] = sy;
    scale[1][2] = yf * (1 - sy);
    scale[2][2] = 1;

    for (int i = 0; i < n; i++) {
        curr[0] = vertices[i].first;
        curr[1] = vertices[i].second;
        curr[2] = 1;
        res = vector<double>(3, 0);
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                res[j] += scale[j][k] * curr[k];
            }
        }
        newvertices.push_back({ round(res[0]),round(res[1]) });
    }

    glBegin(GL_LINE_LOOP);
    glColor3f(200.0f / 255.0f, 100.0f / 255.0f, 100.0f / 255.0f);
    glPointSize(4.0);
    for (int i = 0; i < n; i++) {
        int x = newvertices[i].first;
        int y = newvertices[i].second;
        glVertex2d(x, y);
    }
    glEnd();
}

void reflect_x() {
```

```
    vector<pair<int, int>> newvertices;
    vector<vector<double>> reflect(3, vector<double>(3, 0));
    vector<double> curr(3, 0), res(3, 0);
    reflect[0][0] = 1;
    reflect[1][1] = -1;
    reflect[2][2] = 1;

    for (int i = 0; i < n; i++) {
        curr[0] = vertices[i].first;
        curr[1] = vertices[i].second;
        curr[2] = 1;
        res = vector<double>(3, 0);
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                res[j] += reflect[j][k] * curr[k];
            }
        }
        newvertices.push_back({ round(res[0]),round(res[1]) });
    }

    glBegin(GL_LINE_LOOP);
    glColor3f(200.0f / 255.0f, 100.0f / 255.0f, 100.0f / 255.0f);
    glPointSize(4.0);
    for (int i = 0; i < n; i++) {
        int x = newvertices[i].first;
        int y = newvertices[i].second;
        glVertex2d(x, y);
    }
    glEnd();
}

void reflect_y() {
    vector<pair<int, int>> newvertices;
    vector<vector<double>> reflect(3, vector<double>(3, 0));
    vector<double> curr(3, 0), res(3, 0);
    reflect[0][0] = -1;
    reflect[1][1] = 1;
    reflect[2][2] = 1;

    for (int i = 0; i < n; i++) {
        curr[0] = vertices[i].first;
        curr[1] = vertices[i].second;
        curr[2] = 1;
        res = vector<double>(3, 0);
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                res[j] += reflect[j][k] * curr[k];
            }
        }
        newvertices.push_back({ round(res[0]),round(res[1]) });
    }

    glBegin(GL_LINE_LOOP);
```

```
    glColor3f(200.0f / 255.0f, 100.0f / 255.0f, 100.0f / 255.0f);
    glPointSize(4.0);
    for (int i = 0; i < n; i++) {
        int x = newvertices[i].first;
        int y = newvertices[i].second;
        glVertex2d(x, y);
    }
    glEnd();
}

void reflect_origin() {
    vector<pair<int, int>> newvertices;
    vector<vector<double>> reflect(3, vector<double>(3, 0));
    vector<double> curr(3, 0), res(3, 0);
    reflect[0][0] = -1;
    reflect[1][1] = -1;
    reflect[2][2] = 1;

    for (int i = 0; i < n; i++) {
        curr[0] = vertices[i].first;
        curr[1] = vertices[i].second;
        curr[2] = 1;
        res = vector<double>(3, 0);
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                res[j] += reflect[j][k] * curr[k];
            }
        }
        newvertices.push_back({ round(res[0]),round(res[1]) });
    }

    glBegin(GL_LINE_LOOP);
    glColor3f(200.0f / 255.0f, 100.0f / 255.0f, 100.0f / 255.0f);
    glPointSize(4.0);
    for (int i = 0; i < n; i++) {
        int x = newvertices[i].first;
        int y = newvertices[i].second;
        glVertex2d(x, y);
    }
    glEnd();
}

void reflect_line() {
    vector<pair<int, int>> newvertices;
    vector<vector<double>> reflect(3, vector<double>(3, 0));
    vector<double> curr(3, 0), res(3, 0);
    reflect[0][0] = 1;
    reflect[1][1] = 1;
    reflect[2][2] = 1;

    for (int i = 0; i < n; i++) {
        curr[0] = vertices[i].first;
        curr[1] = vertices[i].second;
```

```
        curr[2] = 1;
        res = vector<double>(3, 0);
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                res[j] += reflect[j][k] * curr[k];
            }
        }
        newvertices.push_back({ round(res[0]),round(res[1]) });
    }

    glBegin(GL_LINE_LOOP);
    glColor3f(200.0f / 255.0f, 100.0f / 255.0f, 100.0f / 255.0f);
    glPointSize(4.0);
    for (int i = 0; i < n; i++) {
        int x = newvertices[i].first;
        int y = newvertices[i].second;
        glVertex2d(x, y);
    }
    glEnd();
}

void shear_x() {
    vector<pair<int, int>> newvertices;
    vector<vector<double>> shear(3, vector<double>(3, 0));
    vector<double> curr(3, 0), res(3, 0);
    shear[0][0] = 1;
    shear[1][1] = 1;
    shear[2][2] = 1;
    shear[0][1] = shx;

    for (int i = 0; i < n; i++) {
        curr[0] = vertices[i].first;
        curr[1] = vertices[i].second;
        curr[2] = 1;
        res = vector<double>(3, 0);
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                res[j] += shear[j][k] * curr[k];
            }
        }
        newvertices.push_back({ round(res[0]),round(res[1]) });
    }

    glBegin(GL_LINE_LOOP);
    glColor3f(200.0f / 255.0f, 100.0f / 255.0f, 100.0f / 255.0f);
    glPointSize(4.0);
    cout << "\nAlong X :";
    for (int i = 0; i < n; i++) {
        int x = newvertices[i].first;
        int y = newvertices[i].second;
        cout << "\n\t" << x << " " << y;

        glVertex2d(x, y);
```

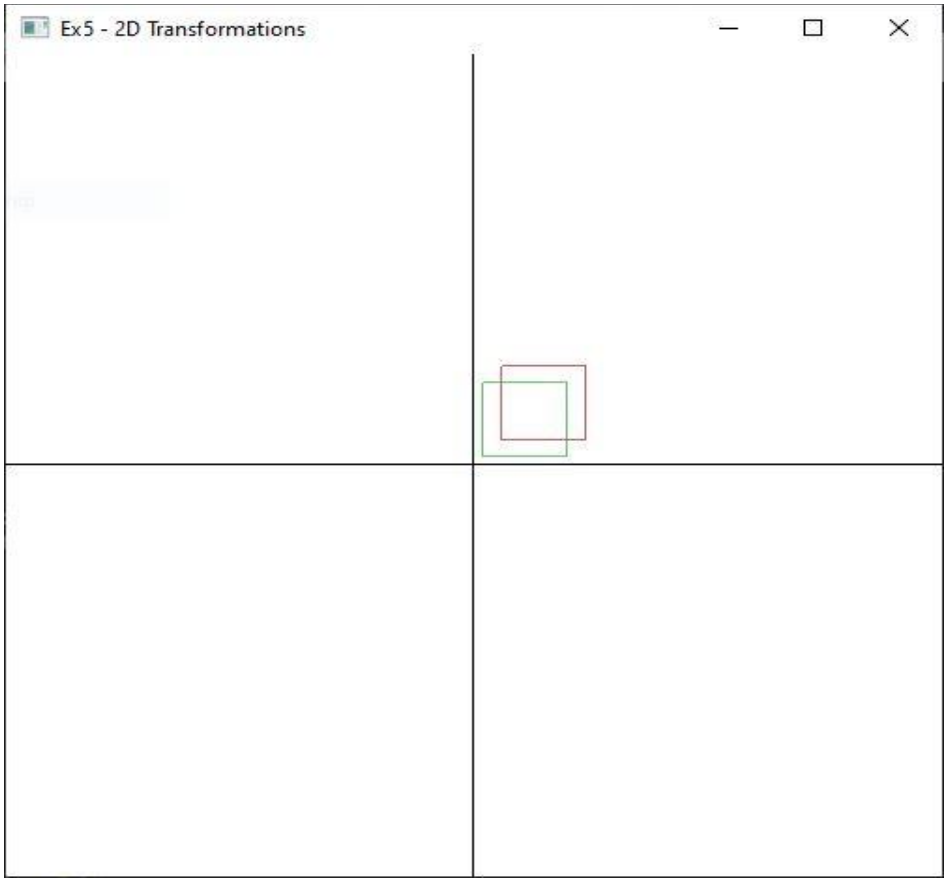
```
    }
    glEnd();
}

void shear_y() {
    vector<pair<int, int>> newvertices;
    vector<vector<double>> shear(3, vector<double>(3, 0));
    vector<double> curr(3, 0), res(3, 0);
    shear[0][0] = 1;
    shear[1][1] = 1;
    shear[2][2] = 1;
    shear[1][0] = shy;

    for (int i = 0; i < n; i++) {
        curr[0] = vertices[i].first;
        curr[1] = vertices[i].second;
        curr[2] = 1;
        res = vector<double>(3, 0);
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                res[j] += shear[j][k] * curr[k];
            }
        }
        newvertices.push_back({ round(res[0]),round(res[1]) });
    }

    glBegin(GL_LINE_LOOP);
    glColor3f(200.0f / 255.0f, 100.0f / 255.0f, 100.0f / 255.0f);
    glPointSize(4.0);
    cout << "\nAlong Y :";
    for (int i = 0; i < n; i++) {
        int x = newvertices[i].first;
        int y = newvertices[i].second;
        cout << "\n\t" << x << " " << y;
        glVertex2d(x, y);
    }
    glEnd();
}

void menu_driven() {
    glClear(GL_COLOR_BUFFER_BIT);
    drawPolygon();

    char sub_opt;
    switch (opt) {
    case 1:

        cout << "\nTranslation factor : ";
        cin >> tx >> ty;
        translation();
        break;

    case 2:
```

```
        cout << "\nAngle of rotation : ";
        cin >> ang;
        angRad = ang * pi / 180;

        cout << "\n\t\ta) about origin ";
        cout << "\n\t\tb) with respect to a fixed point (xr,yr) ";
        cout << "\n\t\tSelect option -> ";

        cin >> sub_opt;
        if (sub_opt == 'a')             rotation_origin();
        else if (sub_opt == 'b') {
            cout << "Rotate about : ";
            cin >> xr >> yr;
            rotation_fixed_pt();
        }
        break;

    case 3:

        cout << "\nScaling factor : ";
        cin >> sx >> sy;

        cout << "\n\t\ta) origin - Uniform Vs Differential Scaling";
        cout << "\n\t\tb) fixed point(xf, yf) ->";
        cout << "\n\t\tSelect option -> ";
        cin >> sub_opt;
        if (sub_opt == 'a')             scaling_origin();
        else if (sub_opt == 'b') {
            cout << "Scale about : ";
            cin >> xf >> yf;
            scaling_fixed_pt();
        }
        break;

    case 4:

        cout << "\n\t\ta) x-axis";
        cout << "\n\t\tb) y - axis";
        cout << "\n\t\tc) origin";
        cout << "\n\t\td) the line x = y";
        cout << "\n\t\tSelect option -> ";
        cin >> sub_opt;

        if (sub_opt == 'a')             reflect_x();
        else if (sub_opt == 'b')  reflect_y();
        else if (sub_opt == 'c')  reflect_origin();
        else if (sub_opt == 'd')  reflect_line();
        break;

    case 5:

        cout << "\nShear amount : ";
```

```
        cin >> shx >> shy;

        cout << "\n\t\ta) x-direction shear";
        cout << "\n\t\tb) y-direction shear";
        cout << "\n\t\tSelect option -> ";
        cin >> sub_opt;
        if (sub_opt == 'a')             shear_x();
        else if (sub_opt == 'b')   shear_y();
        break;
    default: cout << "INVALID INPUT!!";
    }
    glFlush();
}
```

OUTPUT SNAPSHOTS :

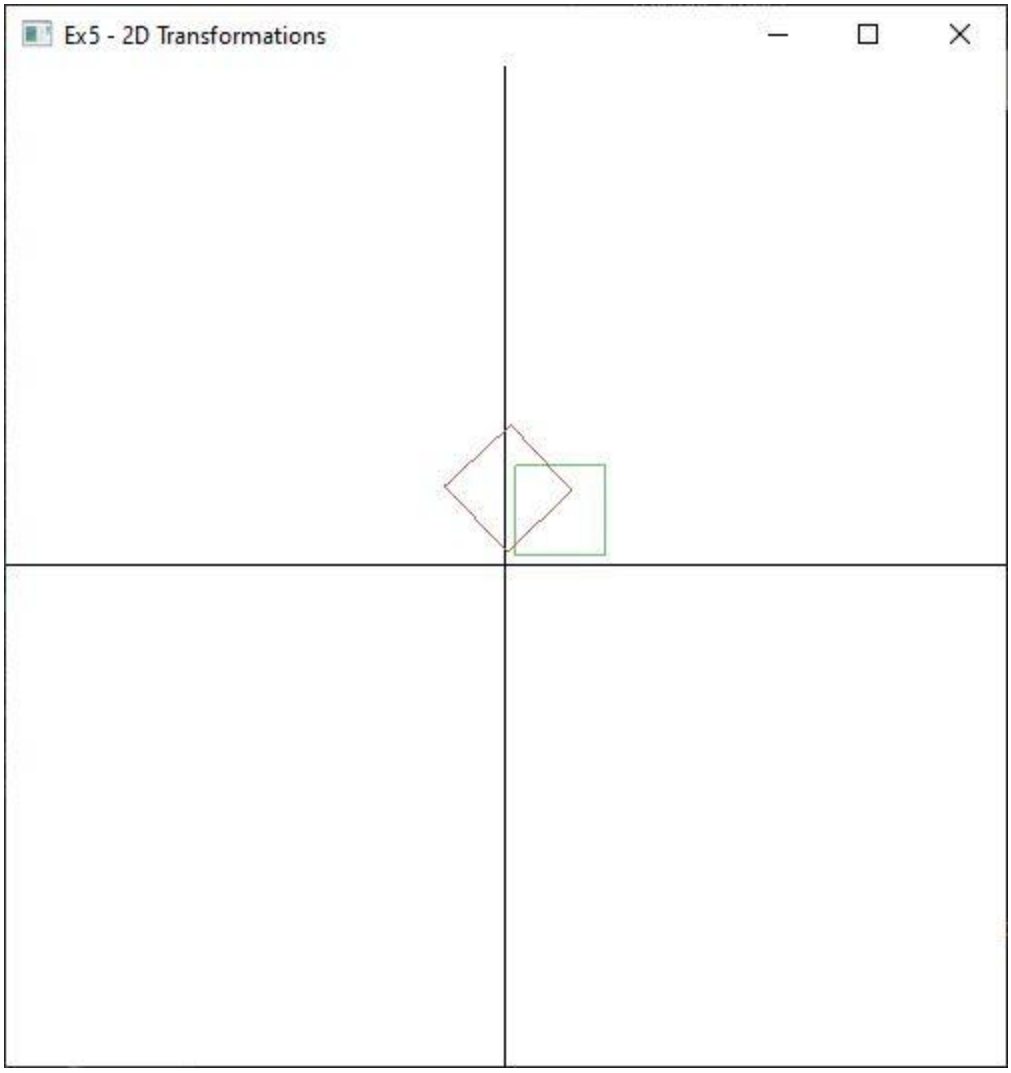
1) Translation :

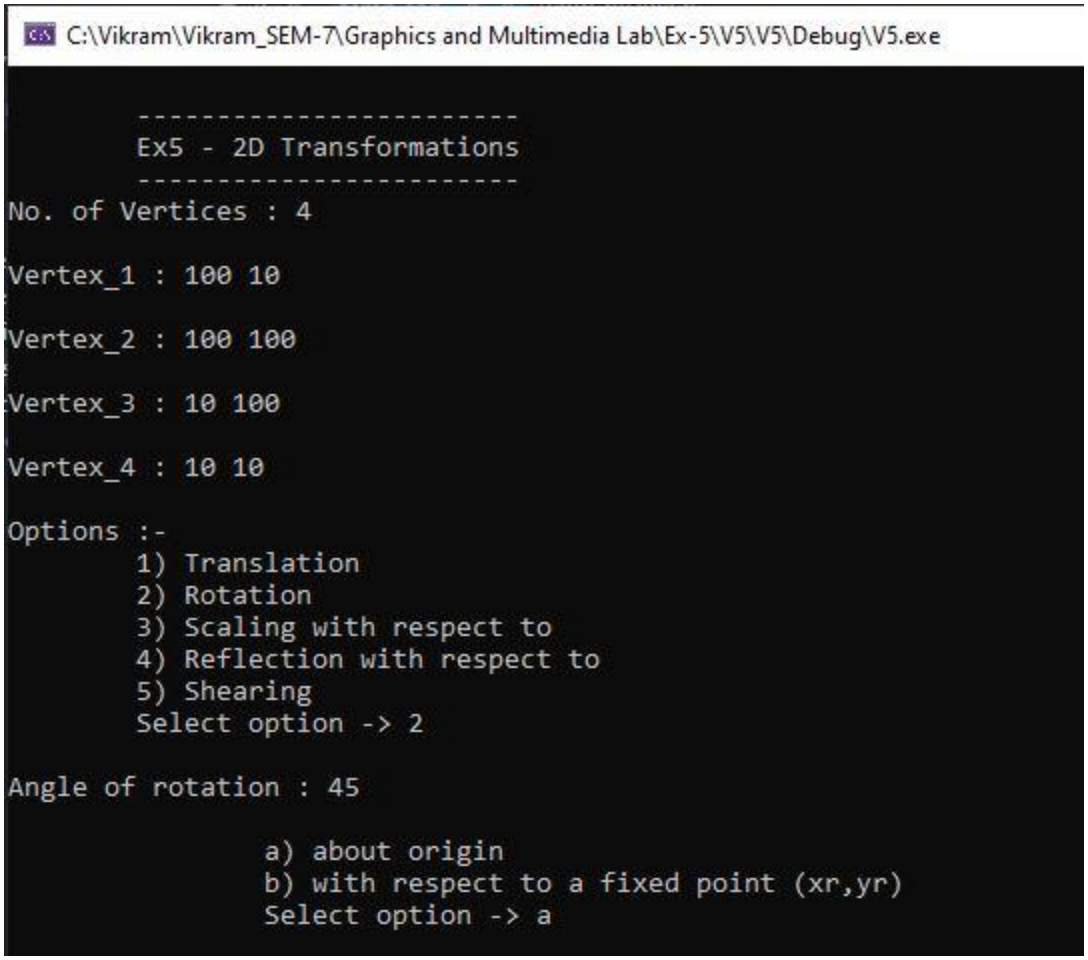



```
C:\Vikram\Vikram_SEM-7\Graphics and Multimedia Lab\Ex-5\V5\V

        ------------------------
        Ex5 - 2D Transformations
        ------------------------
No. of Vertices : 4

Vertex_1 : 10 10

Vertex_2 : 100 10

Vertex_3 : 100 100

Vertex_4 : 10 100
Options :-
        1) Translation
        2) Rotation
        3) Scaling with respect to
        4) Reflection with respect to
        5) Shearing
        Select option -> 1

Translation factor : 20 20
```

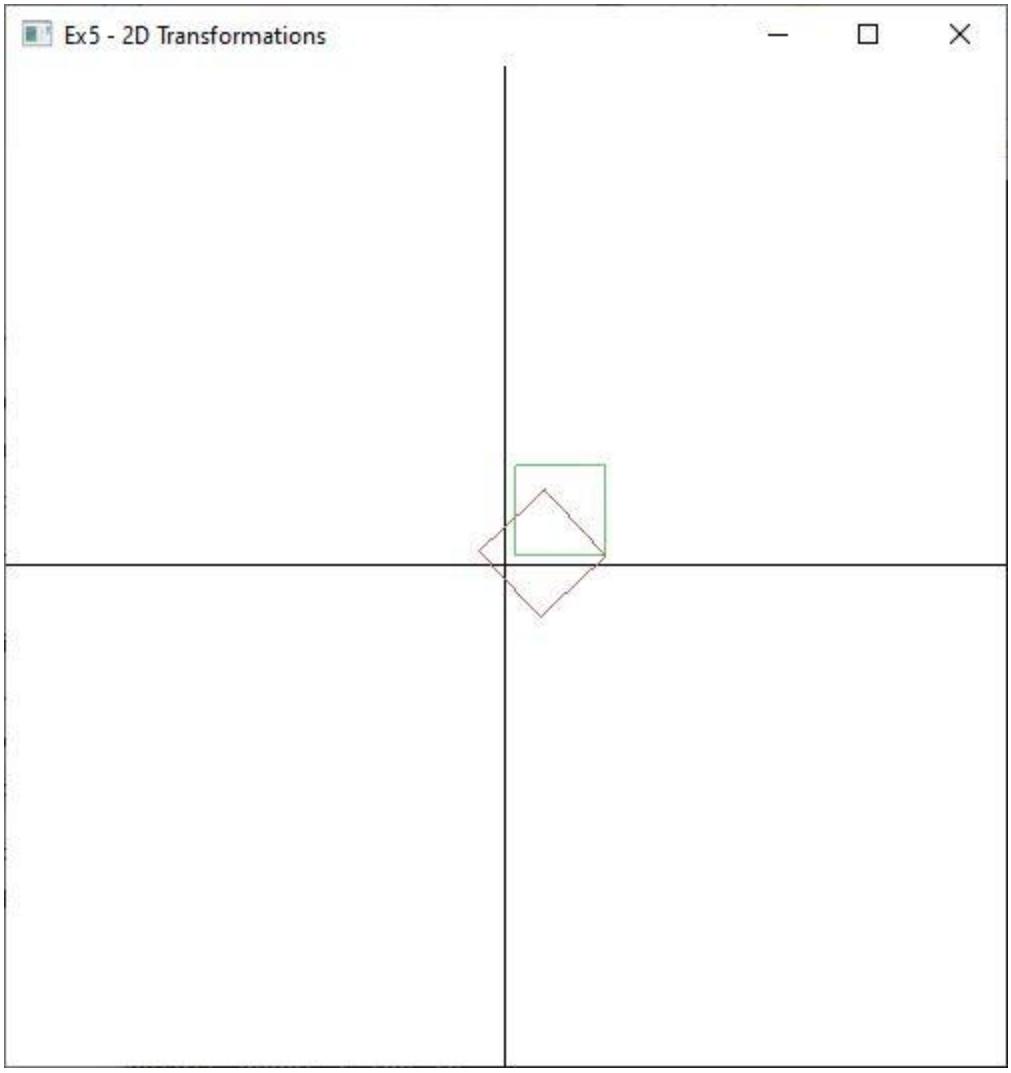
2) Rotation :

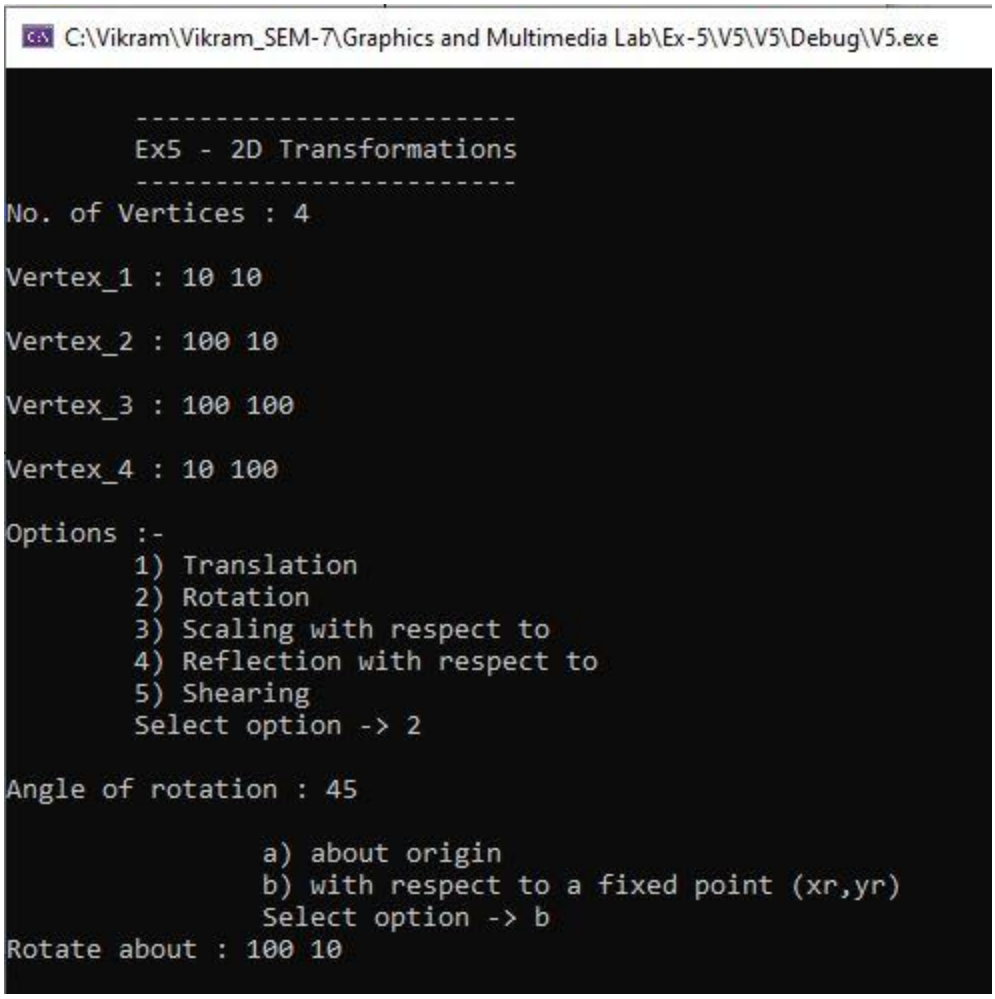
a) about origin :


Ex5 - 2D Transformations



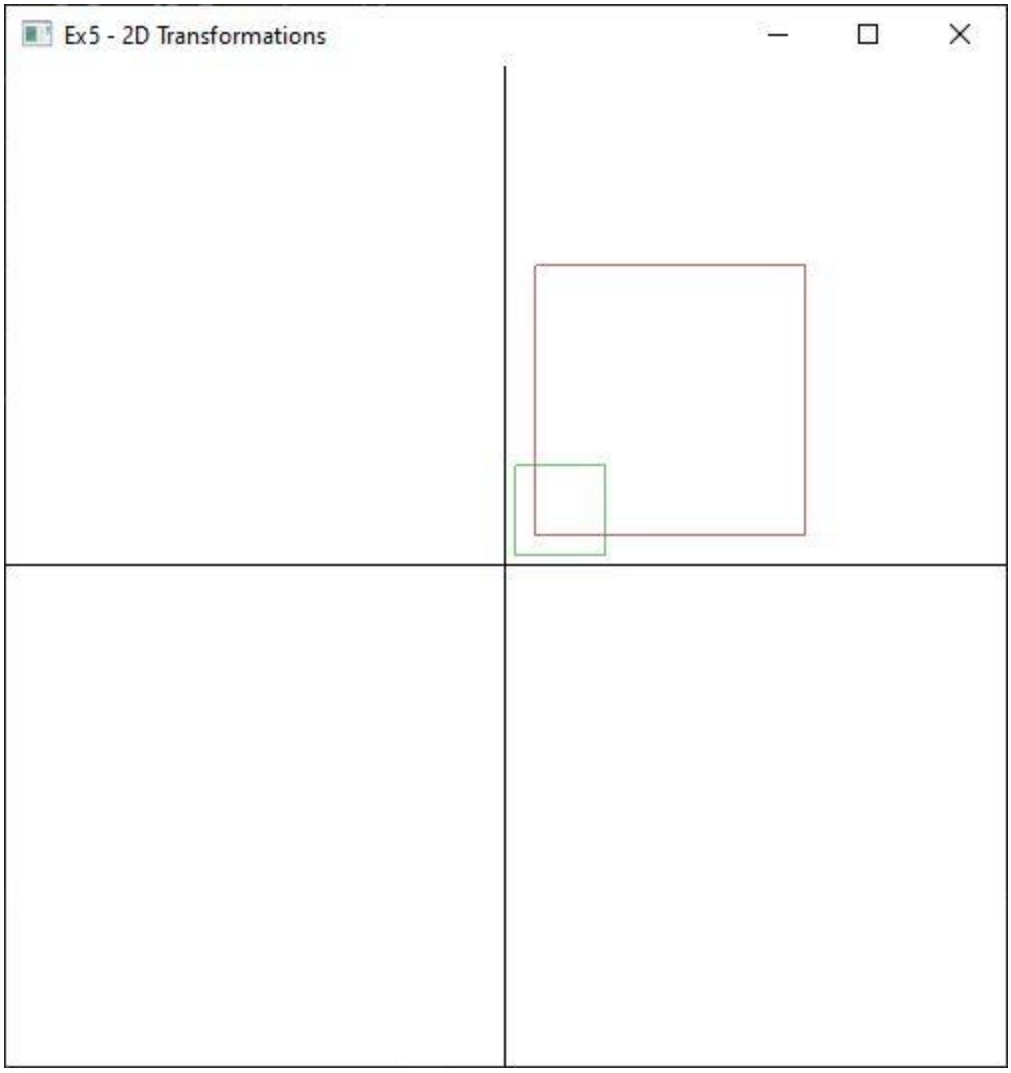
b) with respect to a fixed point (xr,yr) :

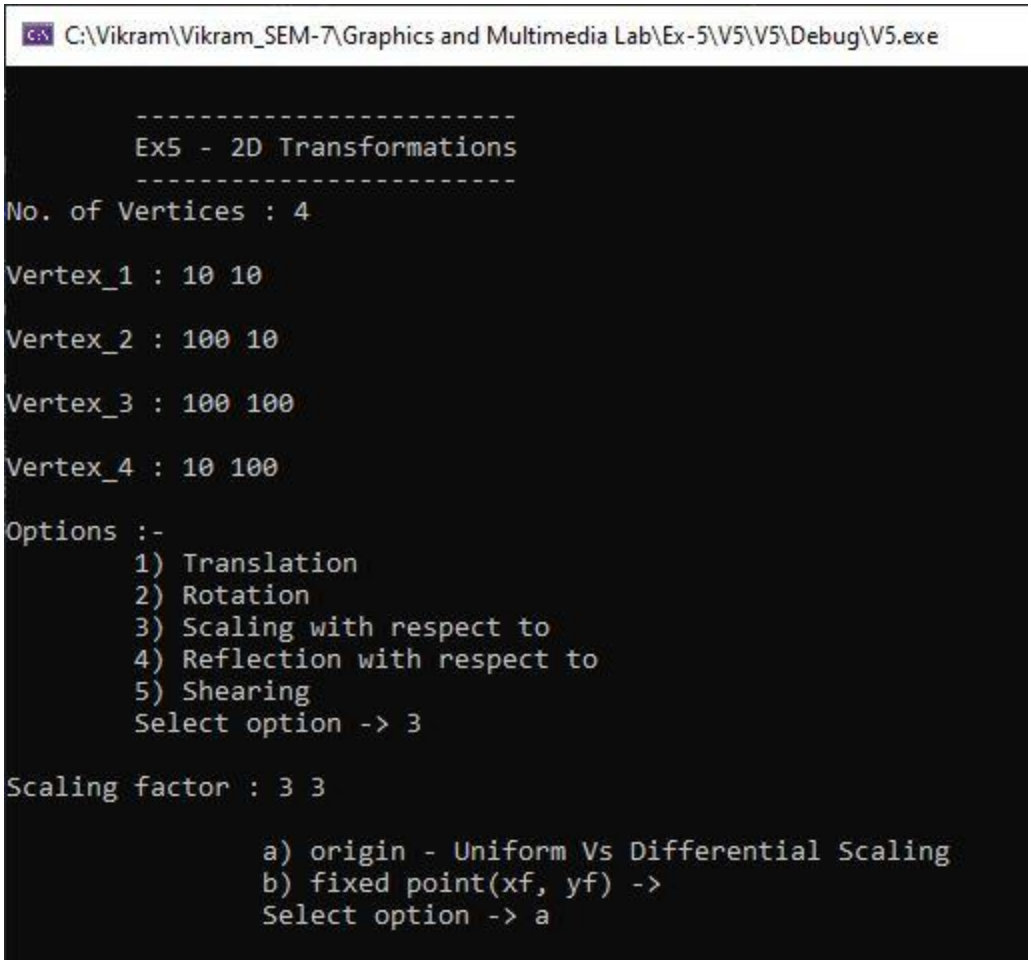

Ex5 - 2D Transformations



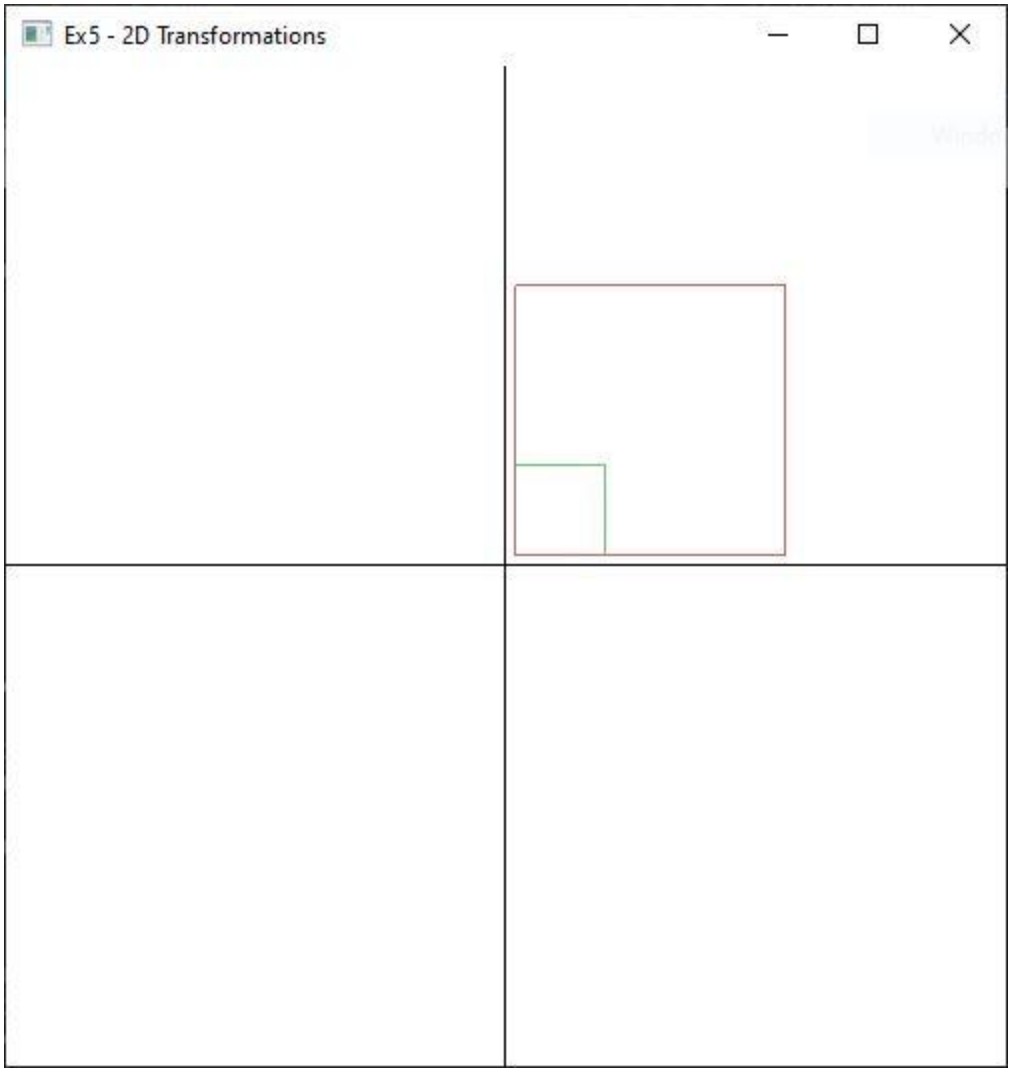
3) Scaling with respect to :

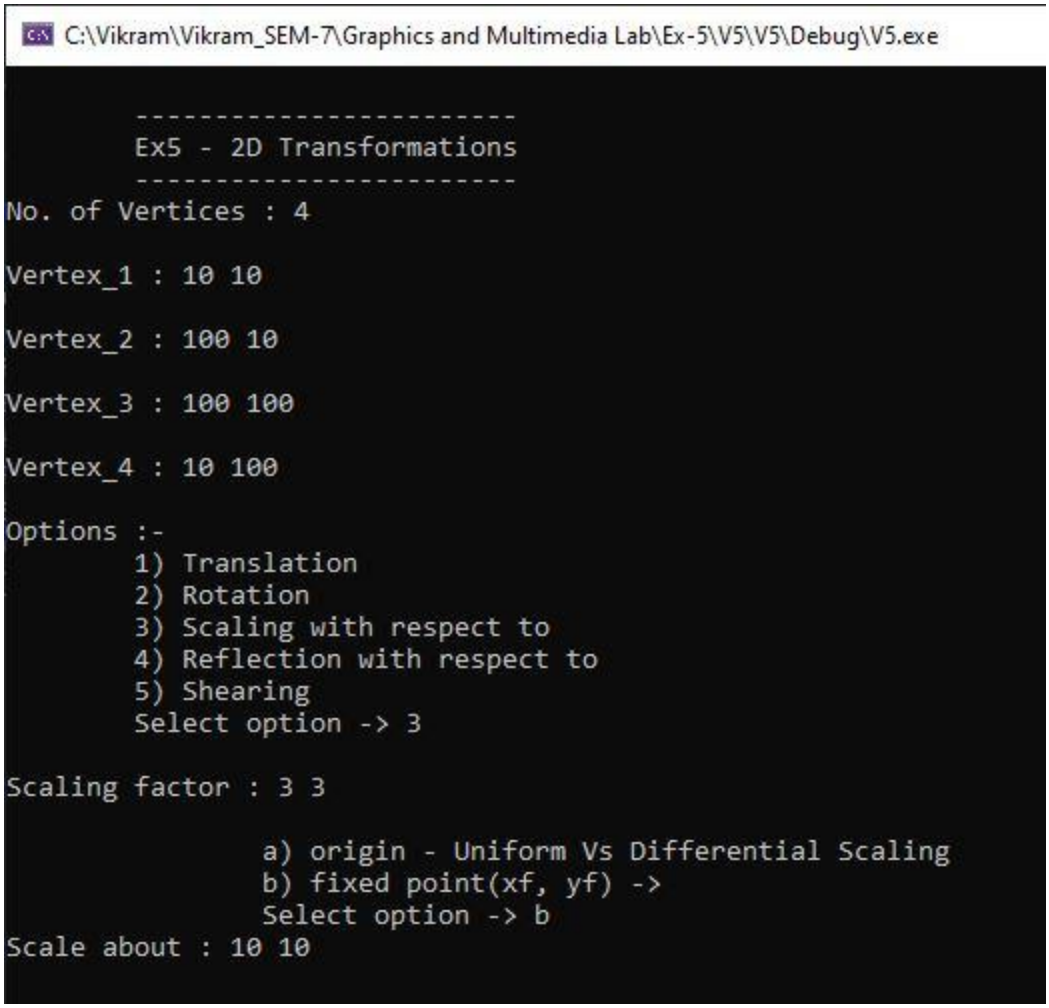
a) origin - Uniform Vs Differential Scaling :


Ex5 - 2D Transformations



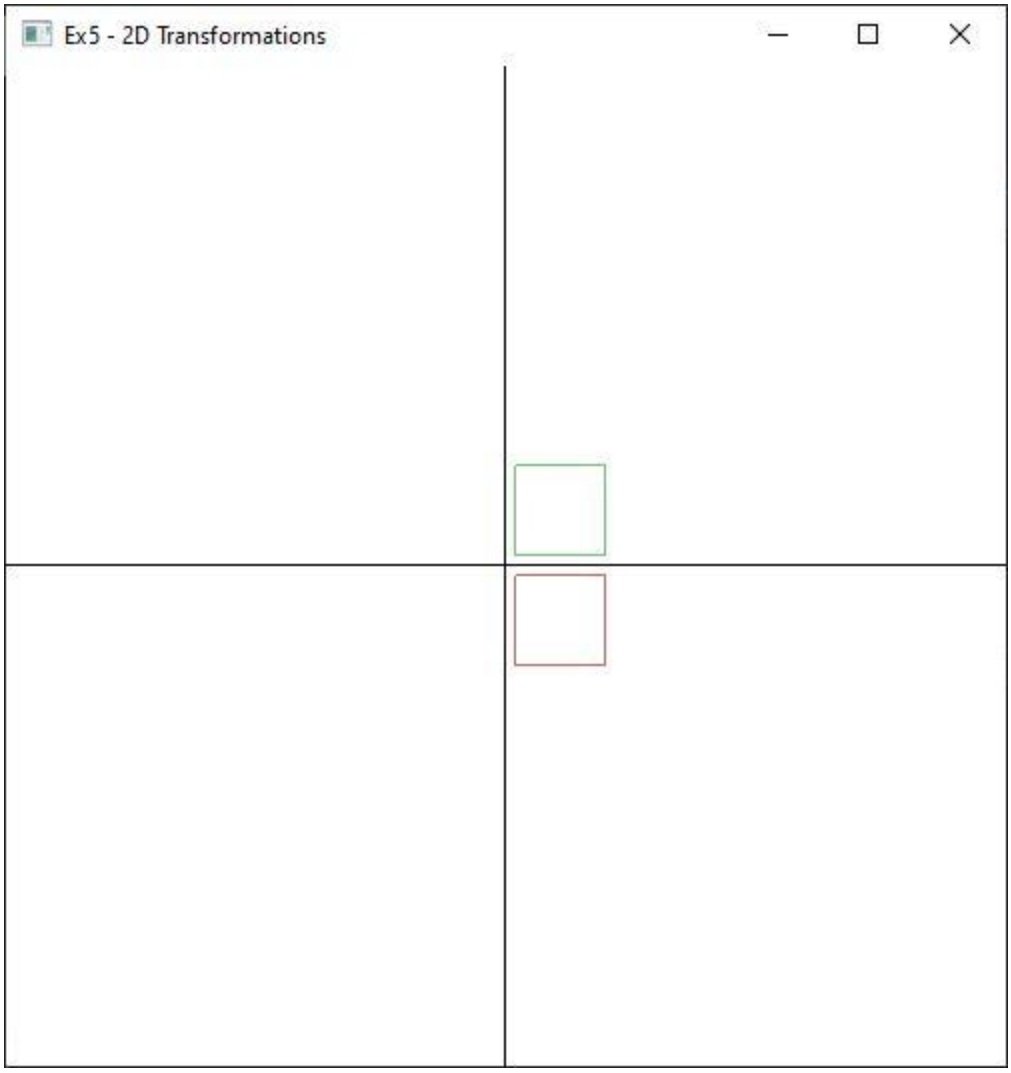
b) fixed point (xf,yf) :

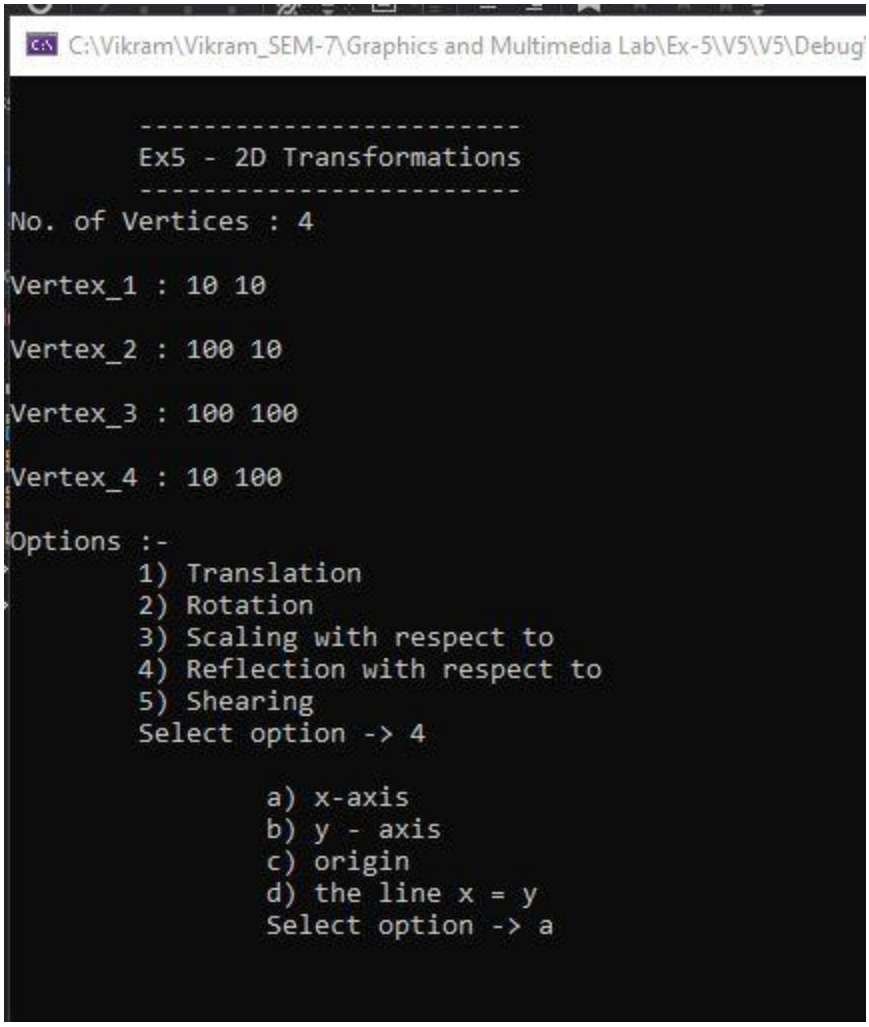

Ex5 - 2D Transformations



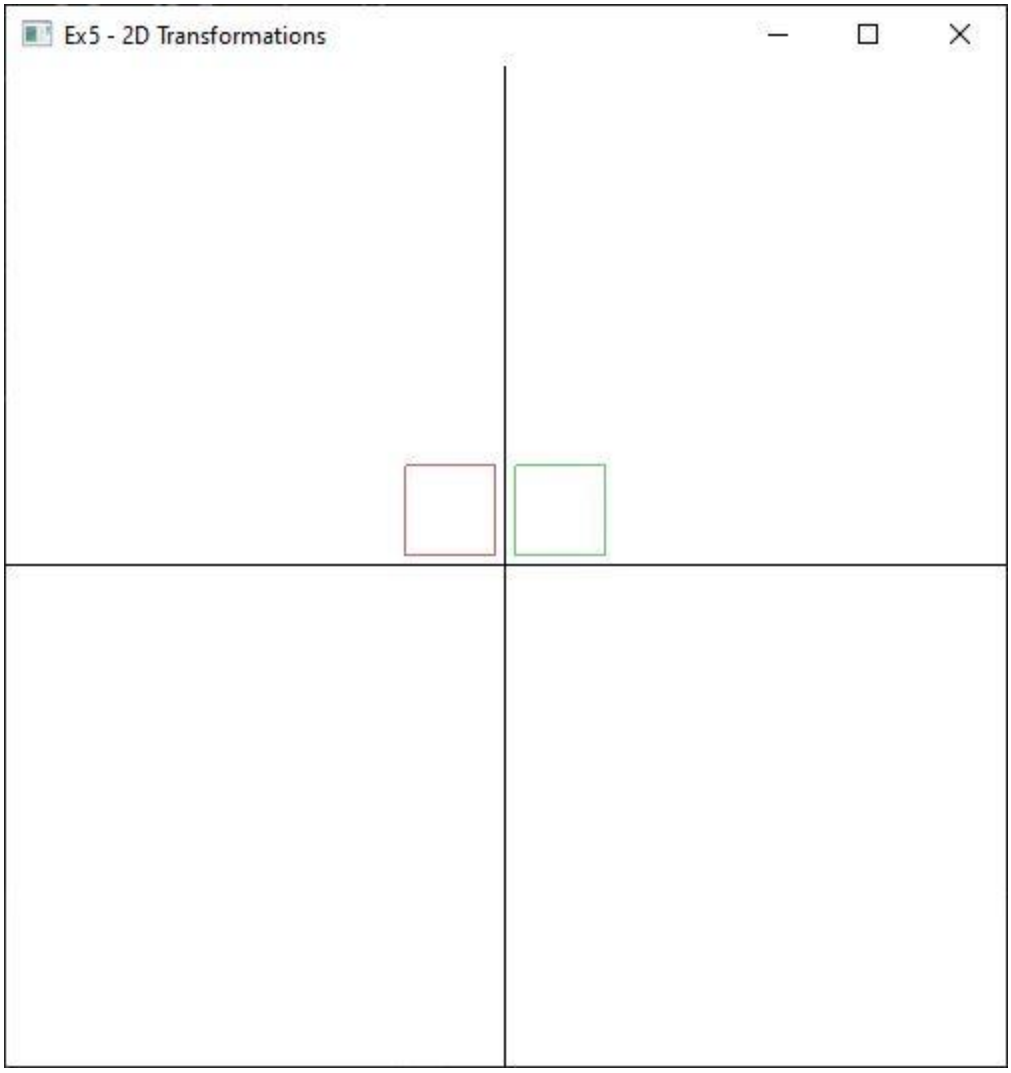
4) Reflection with respect to :

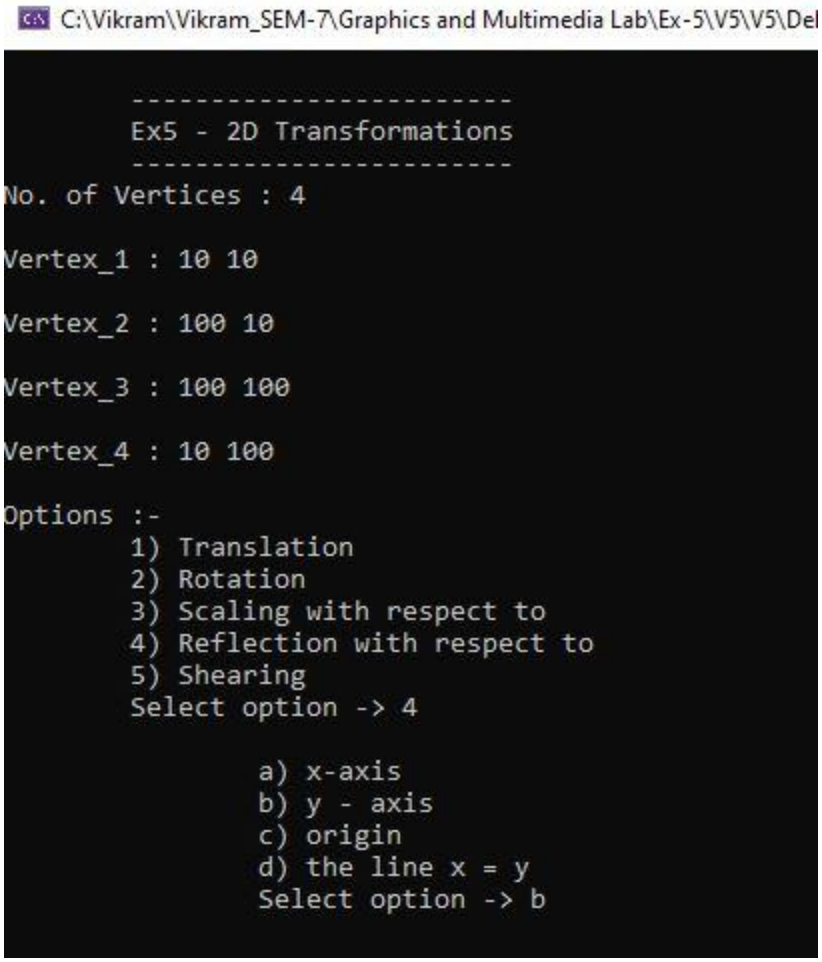
a) x-axis :


Ex5 - 2D Transformations

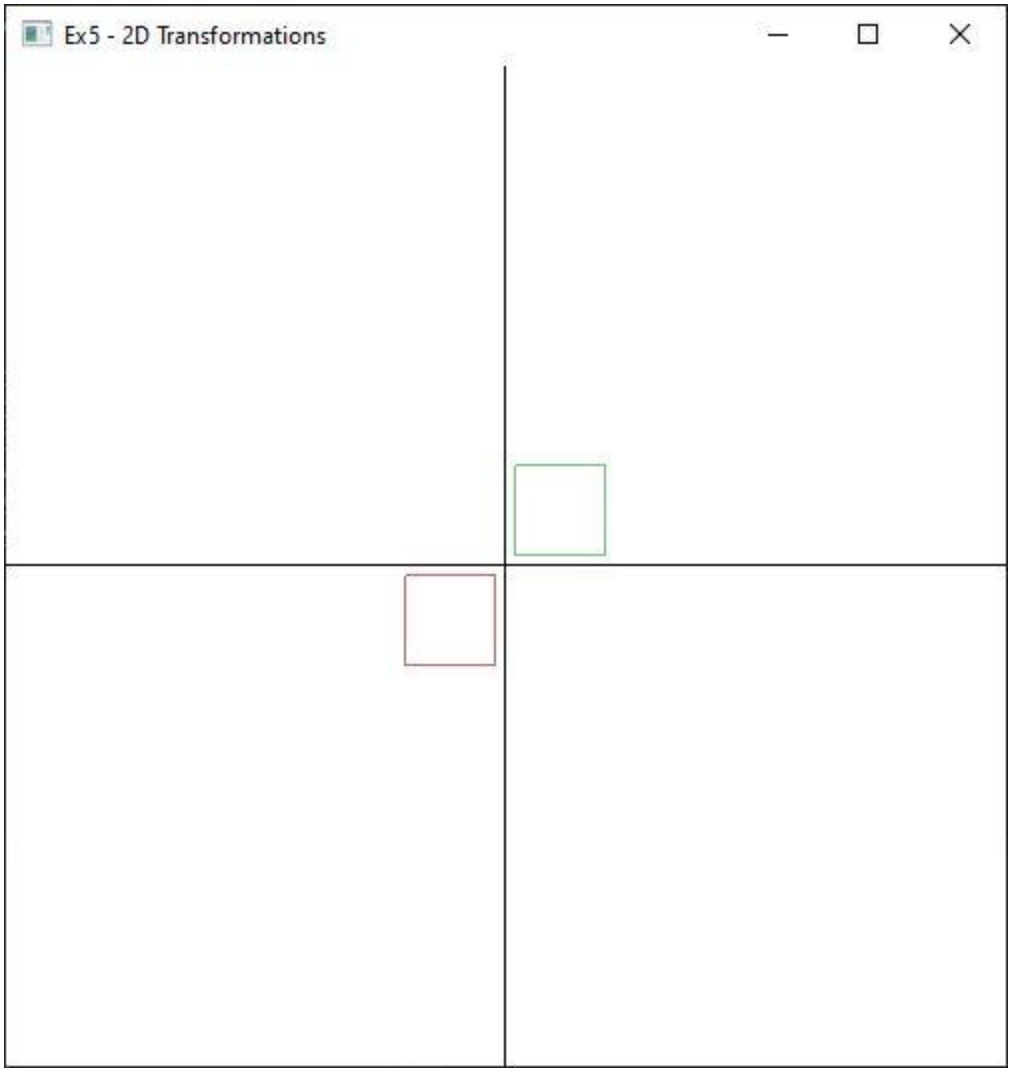

C:\Vikram\Vikram_SEM-7\Graphics and Multimedia Lab\Ex-5\V5\V5\Debug
Ex5 - 2D Transformations
No. of Vertices : 4
Vertex_1 : 10 10
Vertex_2 : 100 10
Vertex_3 : 100 100
Vertex_4 : 10 100
Options :-
1) Translation
2) Rotation
3) Scaling with respect to
4) Reflection with respect to
5) Shearing
Select option -> 4
a) x-axis
b) y - axis
c) origin
d) the line x = y
Select option -> a

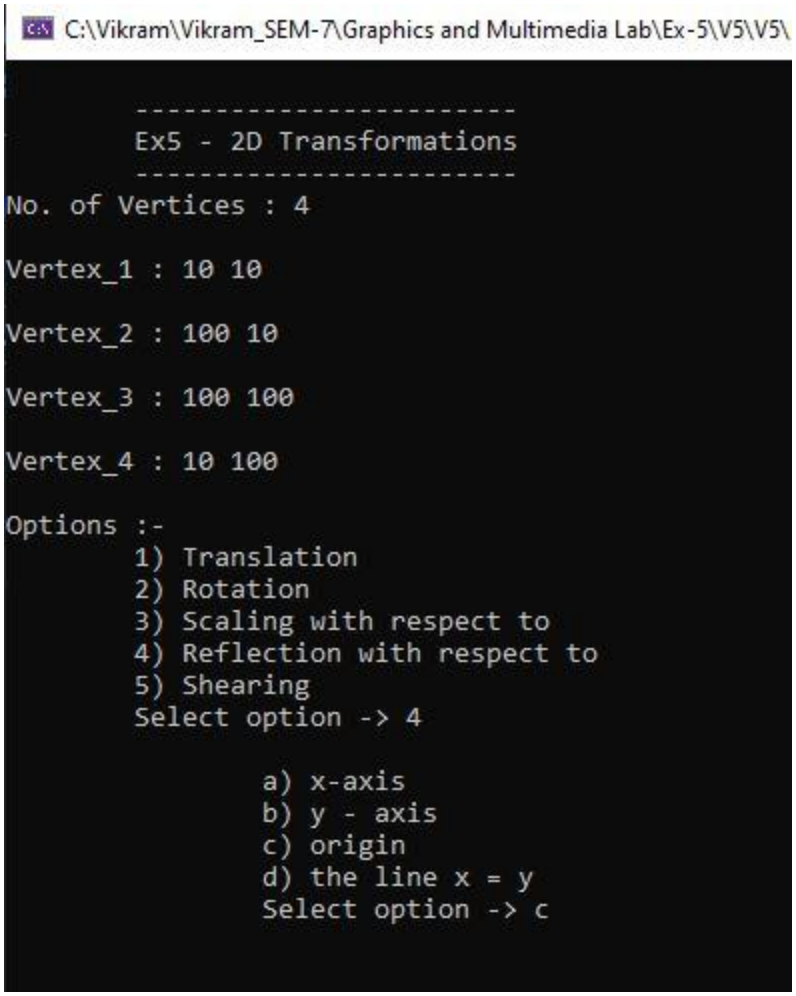
b) y-axis :


Ex5 - 2D Transformations


C:\Vikram\Vikram_SEM-7\Graphics and Multimedia Lab\Ex-5\V5\V5\Del
Ex5 - 2D Transformations
No. of Vertices : 4
Vertex_1 : 10 10
Vertex_2 : 100 10
Vertex_3 : 100 100
Vertex_4 : 10 100
Options :-
1) Translation
2) Rotation
3) Scaling with respect to
4) Reflection with respect to
5) Shearing
Select option -> 4
a) x-axis
b) y - axis
c) origin
d) the line x = y
Select option -> b

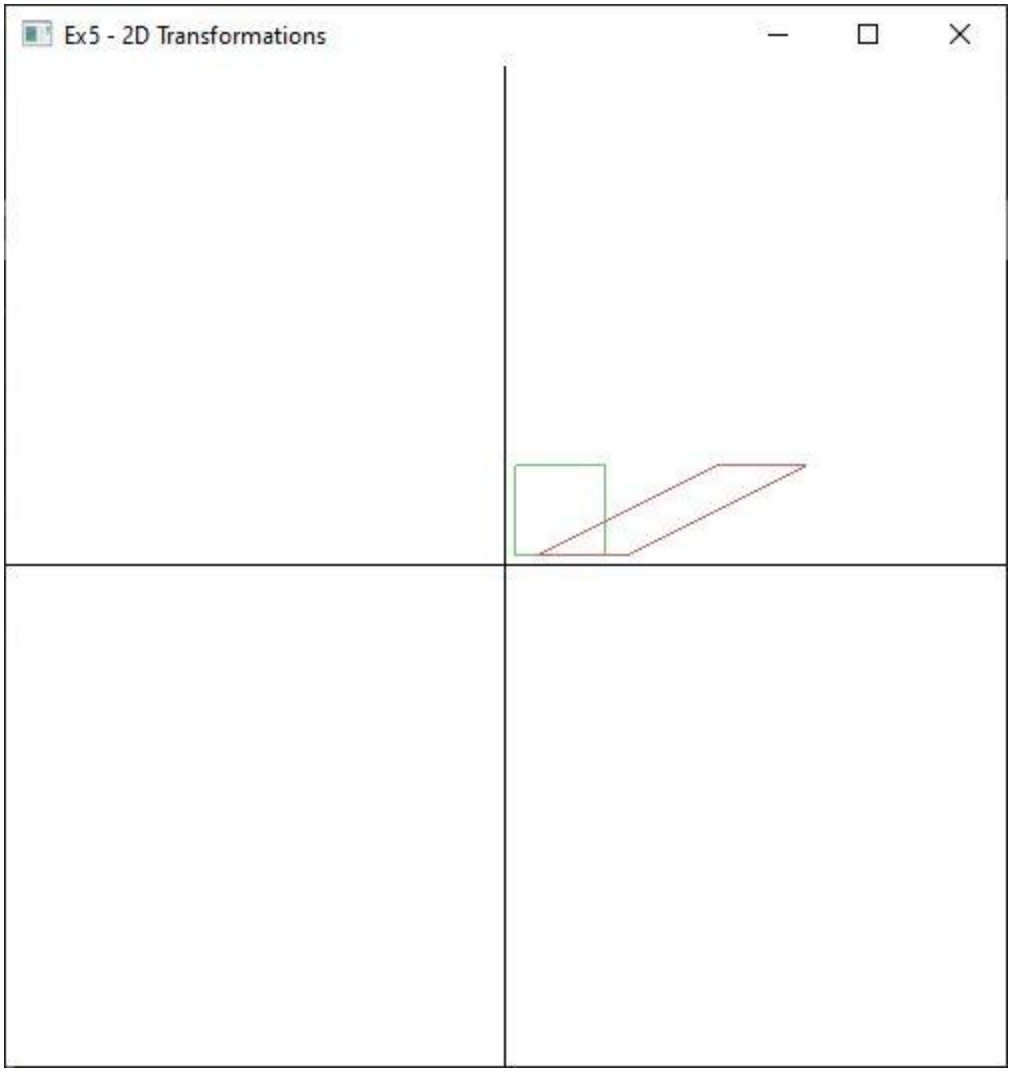
c) origin :

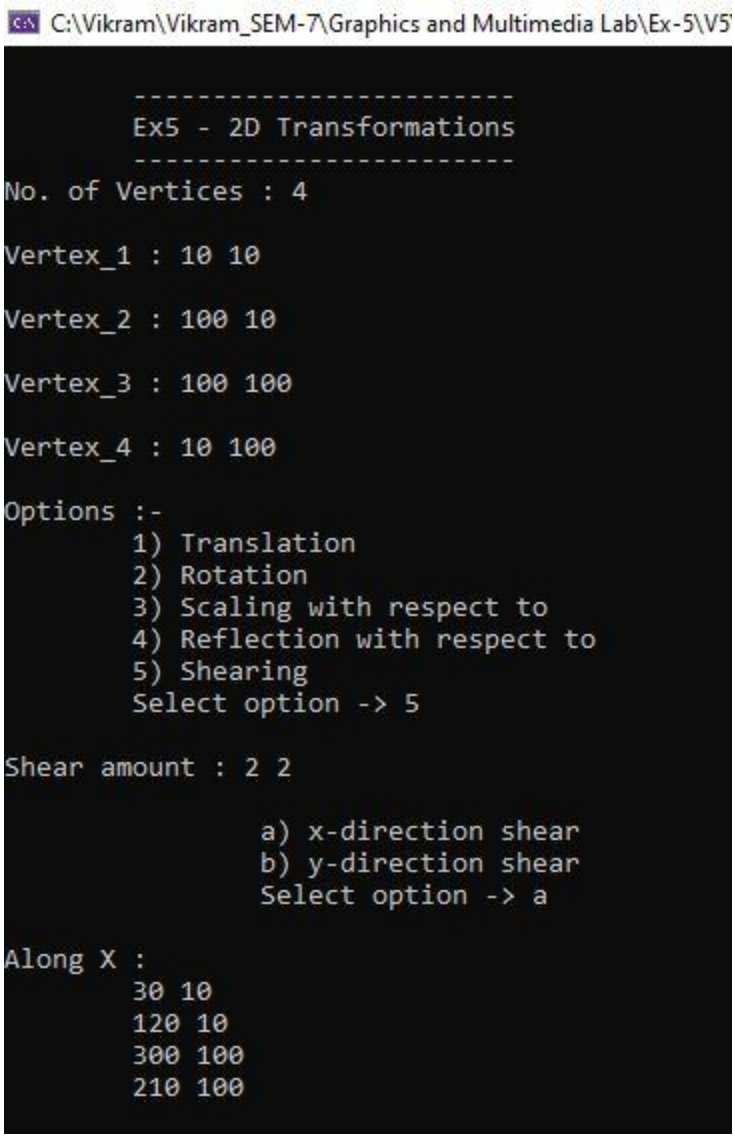

Ex5 - 2D Transformations



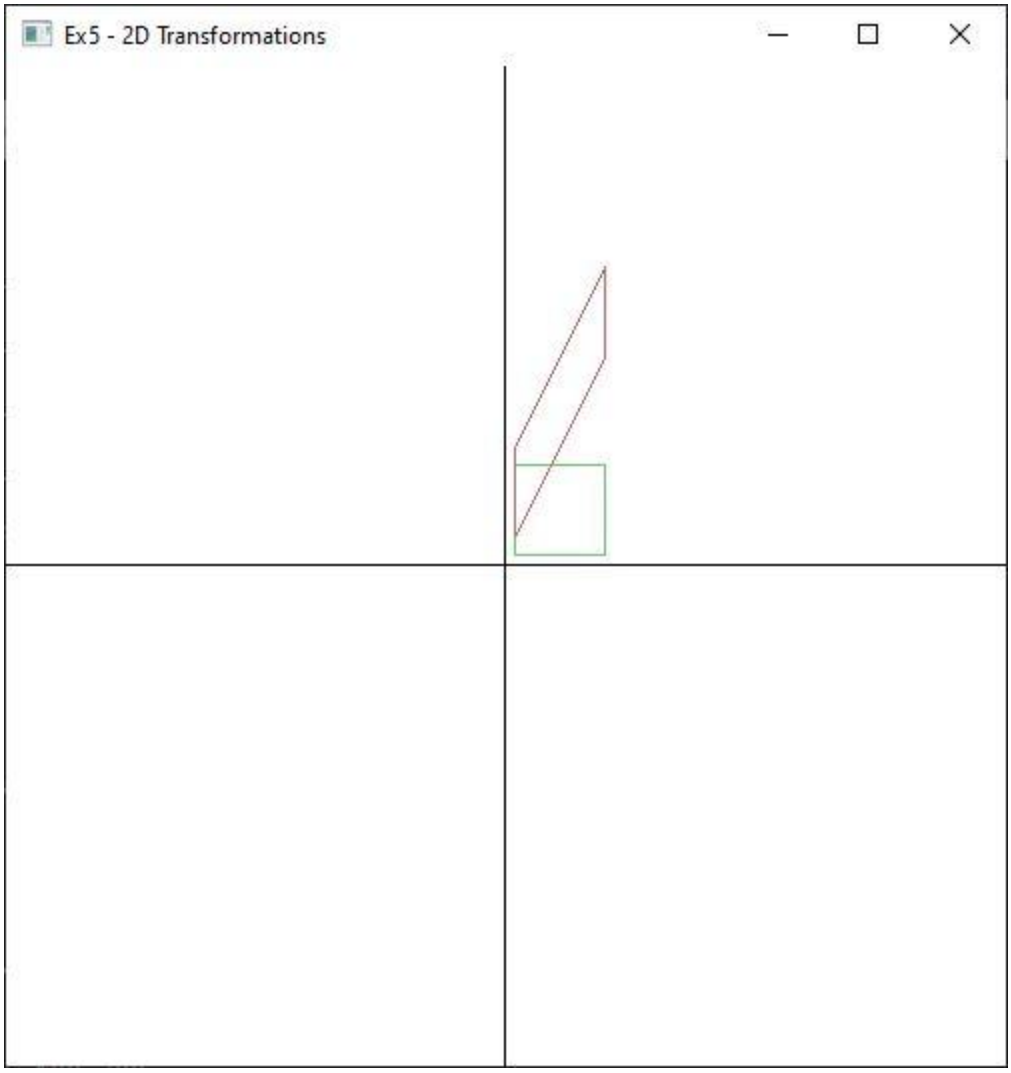
5) Shearing :

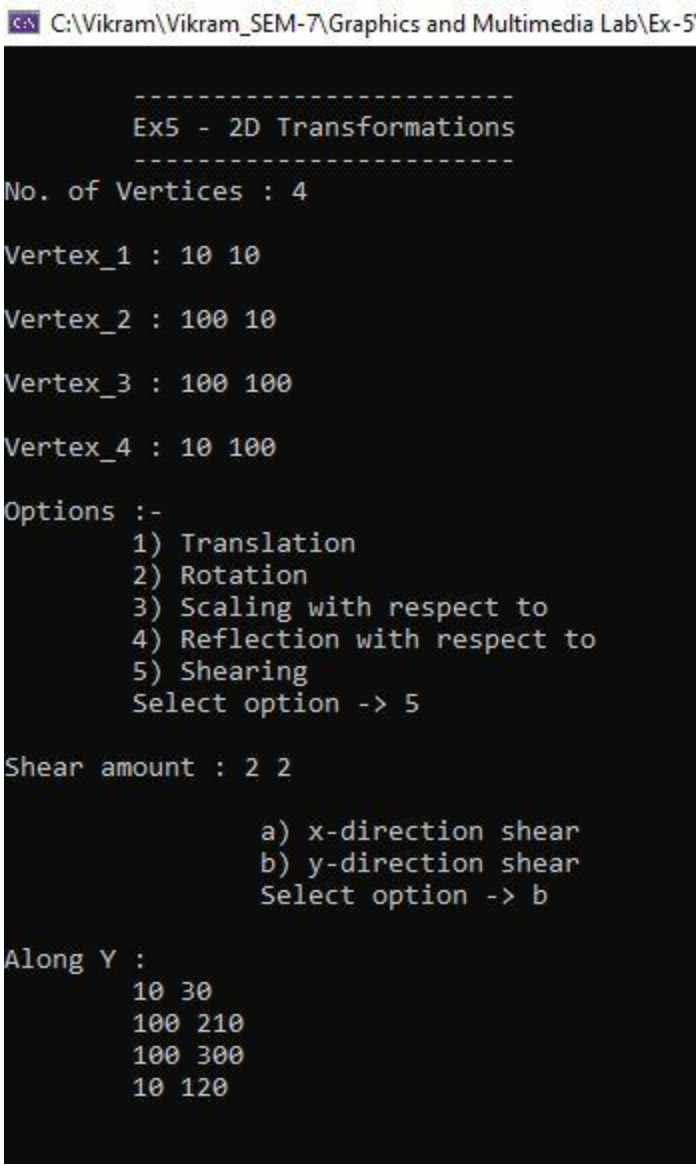
a) x-direction shear :


Ex5 - 2D Transformations



b) y-direction shear:


Ex5 - 2D Transformations



CONCLUSION :

Thus, the 2D transformations on objects were applied and rendered the final output along with the original object.