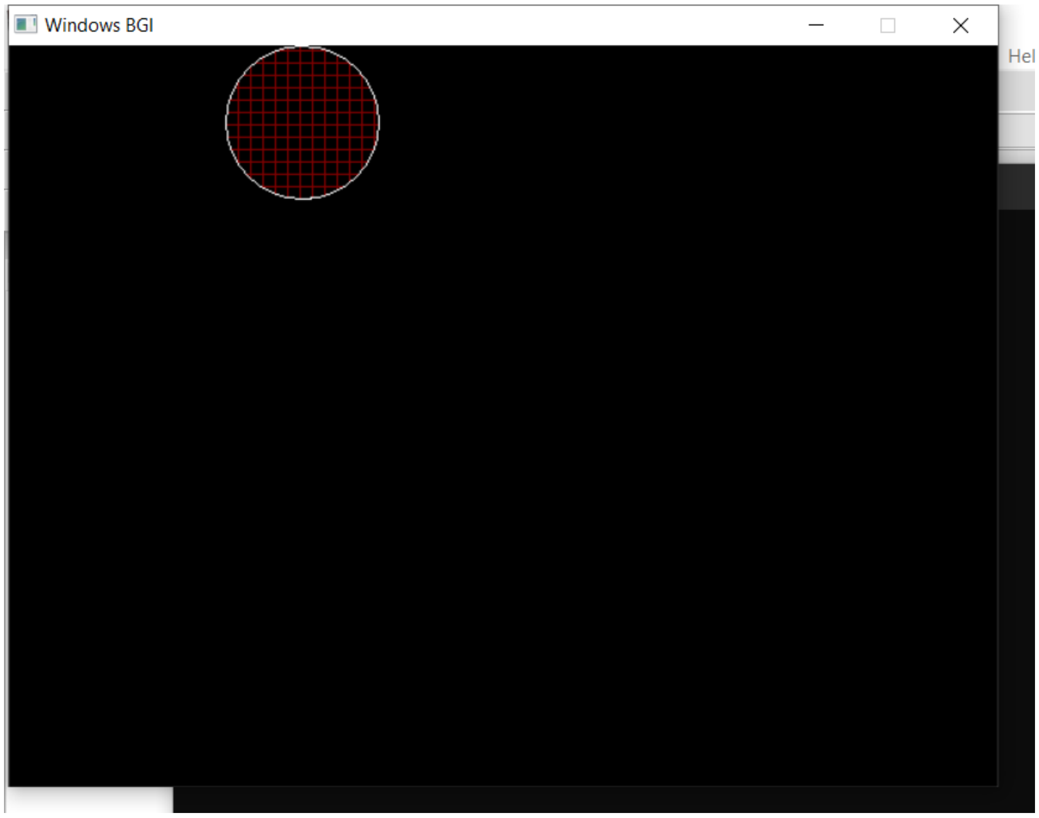
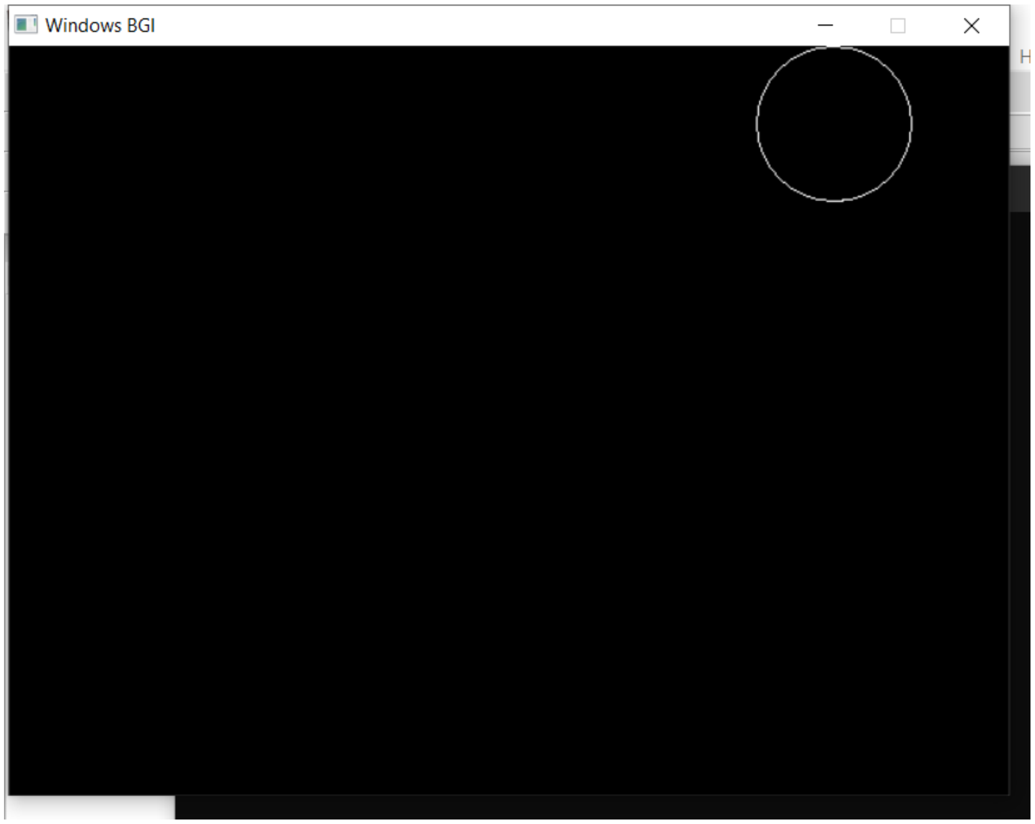
U18CO018
Shubham Shekhaliya
Assignment 7

1 -> A - > Code

```c
#include <stdio.h>
#include <graphics.h>
#include <conio.h>
#include <dos.h>
int main()
{
    int gd = DETECT, gm, i;
    initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
    while (true)
    {
        //for moving circle from left to right,the following loop works
        for (i = 50; i <= getmaxx(); i++)
        {
            setfillstyle(HATCH_FILL, RED);
            circle(50 + i, 50, 50);
            floodfill(52 + i, 52, WHITE);
            delay(25);
            cleardevice();
        }
        //for moving circle from right to left, the following loop works
        for (i = getmaxx(); i >= 0; i--)
        {
            setfillstyle(HATCH_FILL, RED);
            circle(i, 50, 50);
            floodfill(i + 2, 52, WHITE);
            delay(25);
            cleardevice();
        }
    }
    getch();
    return 0;
}
```

Output:-

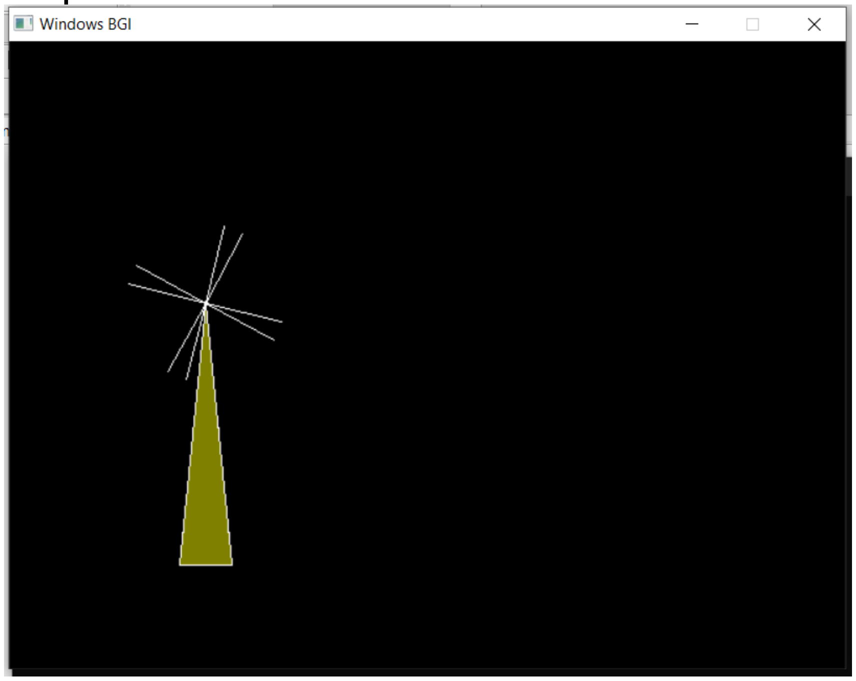

Windows BGI
Hel

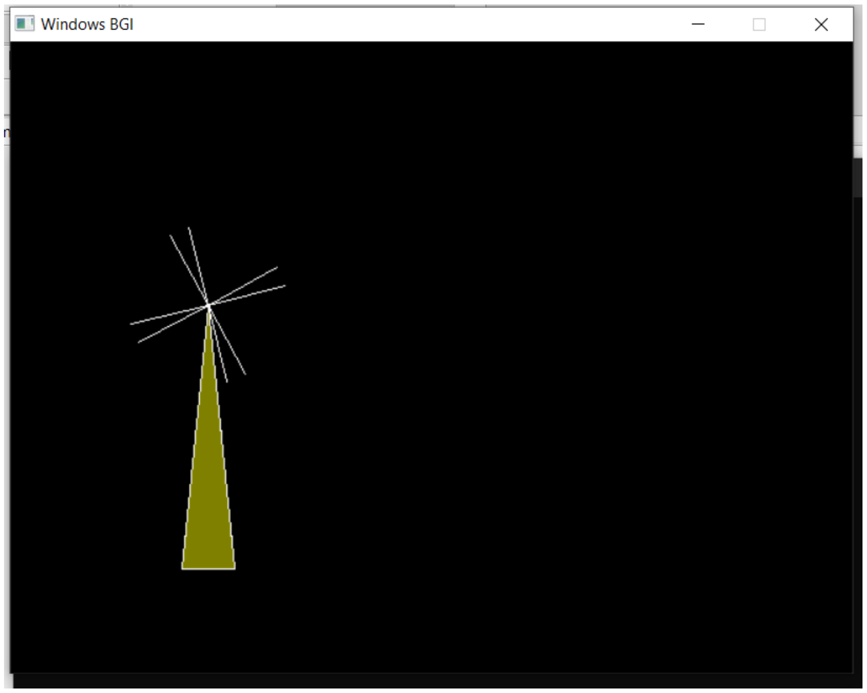

Windows BGI
H

1 - > B -> Code

```
#include <stdio.h>
#include <graphics.h>
#include <math.h>
#include <conio.h>
#include <dos.h>
#define M_PI 3.14
int xc = 150, yc = 200, r = 60;
int x[15], y[15];
int main()
{
    double angle = 0, theta;
    int i, a;
    int gd = DETECT, gm;
    initgraph(&gd, &gm, " ");
    while (!kbhit())
    {
        theta = M_PI * angle / 180;
        cleardevice();
        setfillstyle(SOLID_FILL, BROWN);
        line(150, 200, 130, 400);
        line(150, 200, 170, 400);
        line(130, 400, 170, 400);
        floodfill(152, 398, WHITE);
        for (i = 0; i < 4; i++)
        {
            theta = M_PI * angle / 180;
            x[i] = xc + r * cos(theta);
            y[i] = yc + r * sin(theta);
            angle += 90;
            line(xc, yc, x[i], y[i]);
        }
        angle += 15;
        for (i = 0; i < 4; i++)
        {
            theta = M_PI * angle / 180;
            x[i] = xc + r * cos(theta);
            y[i] = yc + r * sin(theta);
            angle += 90;
            line(xc, yc, x[i], y[i]);
        }
        angle += 15;
        delay(100);
    }
    getch();
    closegraph();
    return 0;
}
```

Output:-

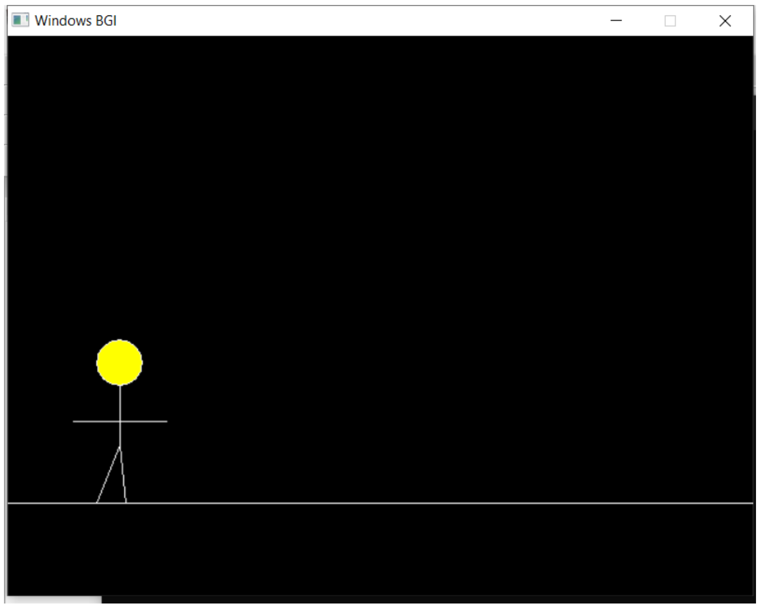


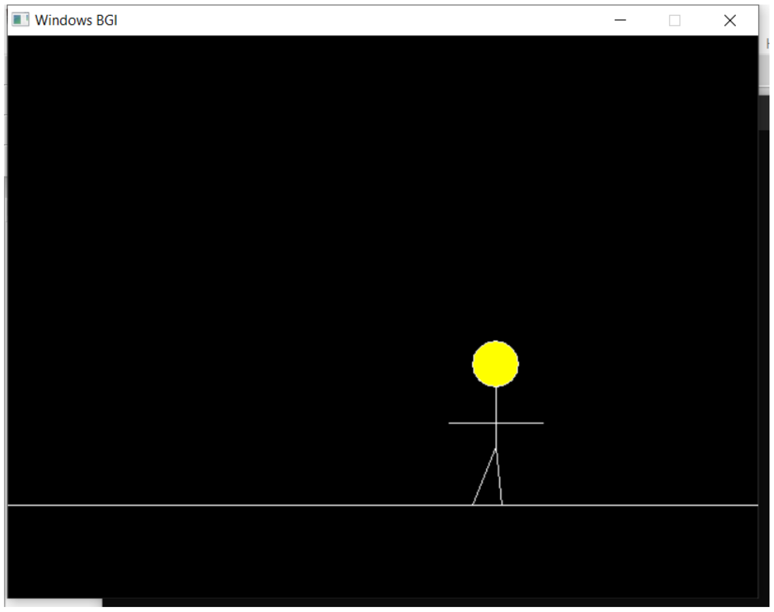


1 -> C -> Code

```c
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <stdlib.h>
int main()
{
    int gr = DETECT, gm;
    int i, x, y, j;
    initgraph(&gr, &gm, "C:\\TURBOC3\\BGI");
    // man
    for (j = 1; j <= getmaxx(); j = j + 5)
    {
        line(0, 400, 800, 400);
        setfillstyle(SOLID_FILL, YELLOW);
        circle(30 + j, 280, 20); //head
        floodfill(32 + j, 280, WHITE);
        line(30 + j, 300, 30 + j, 350);  //body
        line(30 + j, 330, 70 + j, 330);  //right hand
        line(30 + j, 330, -10 + j, 330); //left hand
        if (j % 2 == 0)
        {
            line(30 + j, 350, 35 + j, 400); //left leg
            line(30 + j, 350, 10 + j, 400); // right
        }
        else
        {
            line(30 + j, 350, 35 + j, 400); //transition
            delay(20);
        }
        delay(170);
        cleardevice();
    }
    getch();
    closegraph();
    return 0;
}
```

Output:-

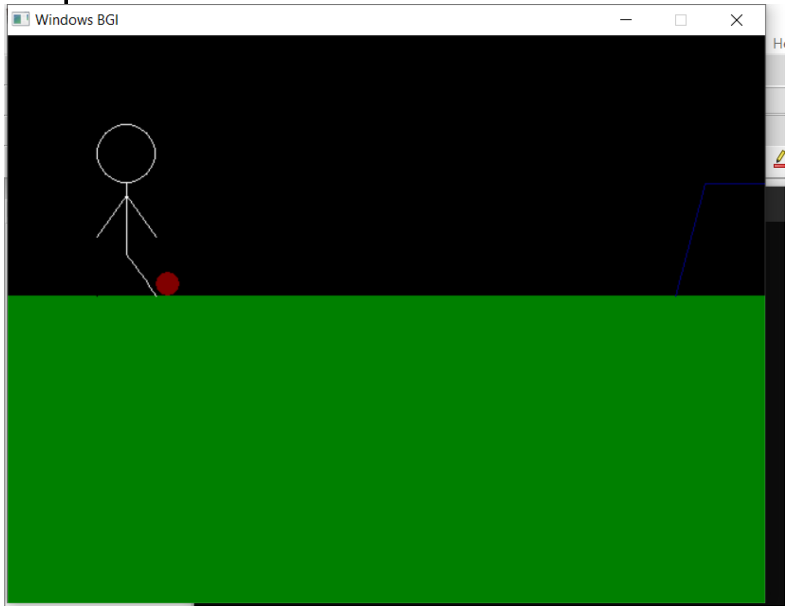


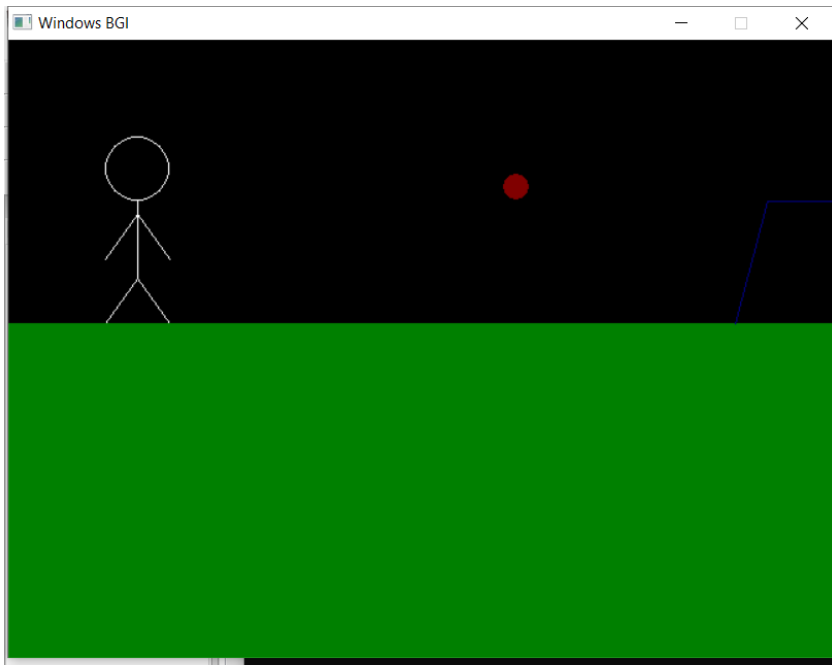


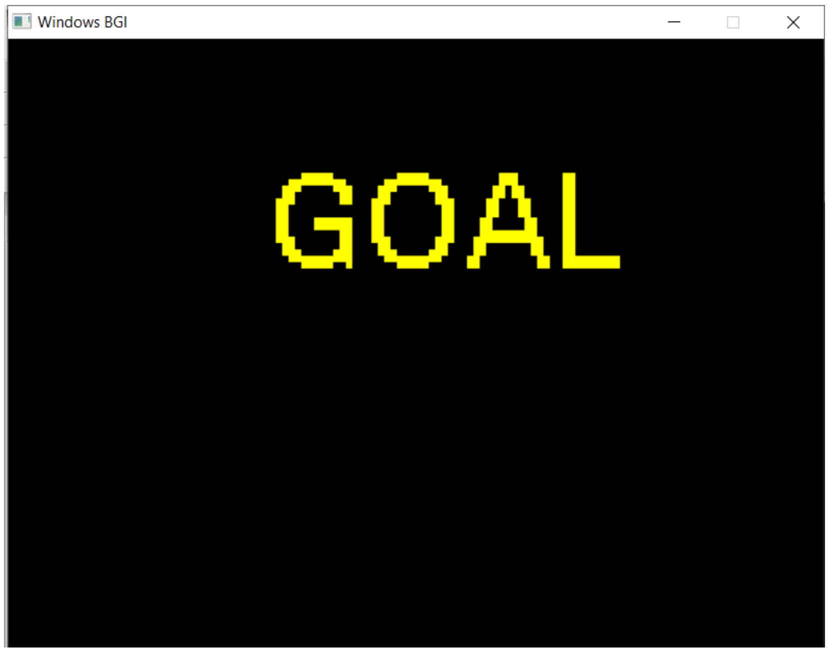
1 -> D -> Code

```c
#include <stdio.h>
#include <graphics.h>
int main()
{
    int x = 0, gd = DETECT, gm, points[] = {0, 220, 1600, 220, 1600, 900, 0, 900,
 0, 220};
    float y = 0;
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
    setcolor(GREEN);
    setfillstyle(SOLID_FILL, GREEN);
    fillpoly(5, points);
    setcolor(WHITE);
    circle(100, 100, 25);
    line(100, 125, 100, 185);
    line(100, 135, 125, 170);
    line(100, 135, 75, 170);
    line(100, 185, 125, 220);
    line(100, 185, 75, 220);
    setcolor(BLUE);
    line(getmaxx(), 125, getmaxx() - 50, 125);
    line(getmaxx() - 50, 125, getmaxx() - 75, 220);
    setcolor(RED);
    setfillstyle(SOLID_FILL, RED);
    fillellipse(135 + x, 210 - y, 10, 10);
    for (x = 0; x < 50; x++)
    {
        setcolor(WHITE);
        line(100, 185, 75 + x, 220 - y);
        delay(50);
        setcolor(BLACK);
        line(100, 185, 75 + x, 220 - y);
        y = y + 0.25;
    }
    setcolor(WHITE);
    line(100, 185, 125, 220);
    line(100, 185, 75, 220);
    for (x = 0, y = 0; y < 100; x++)
    {
        setcolor(RED);
        setfillstyle(SOLID_FILL, RED);
        fillellipse(135 + x, 210 - y, 10, 10);
        delay(10);
        setcolor(BLUE);
        line(getmaxx(), 125, getmaxx() - 50, 125);
        line(getmaxx() - 50, 125, getmaxx() - 75, 220);
        setcolor(GREEN);
        setfillstyle(SOLID_FILL, GREEN);
        fillpoly(5, points);
        setcolor(BLACK);
        setfillstyle(SOLID_FILL, BLACK);
        fillellipse(135 + x, 210 - y, 10, 10);
        y = y + 0.5;
    }
    for (; x < 490; x++)
    {
        setcolor(RED);
        setfillstyle(SOLID_FILL, RED);
        fillellipse(135 + x, y, 10, 10);
        setcolor(BLUE);
        line(getmaxx(), 125, getmaxx() - 50, 125);
        line(getmaxx() - 50, 125, getmaxx() - 75, 220);
        delay(10);
        setcolor(BLACK);
        setfillstyle(SOLID_FILL, BLACK);
        fillellipse(135 + x, y, 10, 10);
        y = y + 0.25;
    }
    setcolor(RED);
    setfillstyle(SOLID_FILL, RED);
    fillellipse(135 + x, y, 10, 10);
    delay(2000);
    cleardevice();
    setcolor(YELLOW);
    settextstyle(3, HORIZ_DIR, 10);
    outtextxy(200, 80, "GOAL");
    getch();
    closegraph();
    return 0;
}
```

Output :-







GOAL

2-> Code

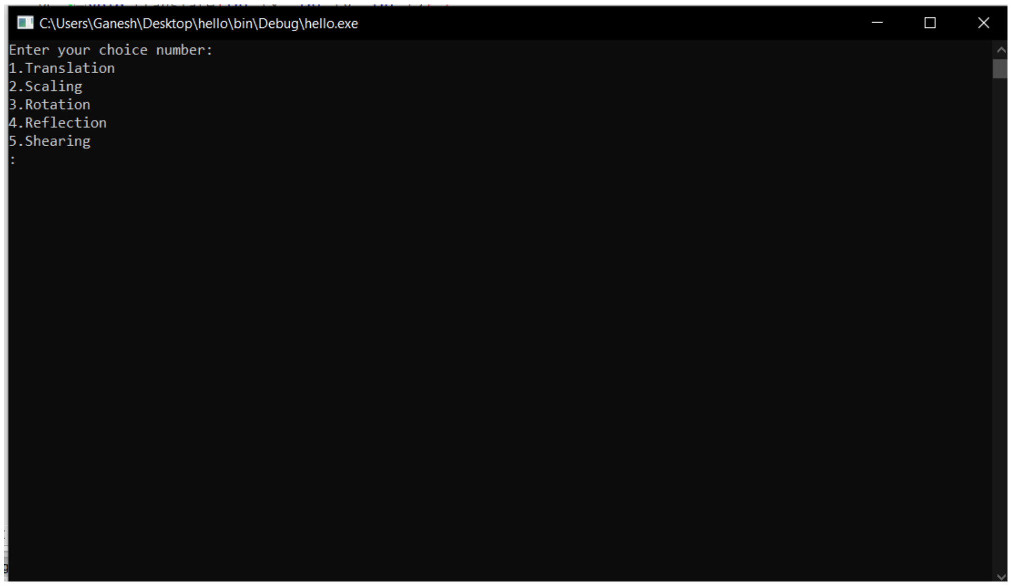
```
#include <GL/glut.h>
#include <iostream>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <unordered_set>
using namespace std;
```

```
typedef float Matrix4x4[4][4];
Matrix4x4 theMatrix;
float ptsIni[8][3] = {{80, 80, -100}, {180, 80, -100}, {180, 180, -
100}, {80, 180, -100}, {60, 60, 0}, {160, 60, 0}, {160, 160, 0}, {60, 160, 0}};
// Realign above line while execution
// Initial Co-ordinates ofthe Cube to be Transformed
int flag = 0;
float ptsFin[8][3];
float refptX, refptY, refptZ;              // Reference points
float TransDistX, TransDistY, TransDistZ; // Translations along Axes
float ScaleX, ScaleY, ScaleZ;              // Scaling Factors along Axes
float Alpha, Beta, Gamma, Theta;           // Rotation angles about Axes
float A, B, C;                             // Arbitrary Line Attributes
float aa, bb, cc;
float shx, shy, shz; // Arbitrary Line Attributes
float x1, y11, z1, x2, y2, z2;
int choiceRot, choiceRef, choiceSh;
unordered_set<int> choice;
void matrixSetIdentity(Matrix4x4 m) // Initialises the matrix as Unit Matrix
{
    int i, j;
    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            m[i][j] = (i == j);
}
void matrix_pre_multiply(Matrix4x4 a, Matrix4x4 b)
{ // Multiplies matrix a times b, putting result in b
    int i, j;
    Matrix4x4 tmp;
    for (i = 0; i < 4; i++)
    {
        for (j = 0; j < 4; j++)
        {
            tmp[i][j] = a[i][0] * b[0][j] + a[i][1] * b[1][j] + a[i][2] * b[2][j]
 +
                        a[i][3] * b[3][j];
        }
    }
    for (i = 0; i < 4; i++)
    {
        for (j = 0; j < 4; j++)
        {
            theMatrix[i][j] = tmp[i][j];
        }
    }
```

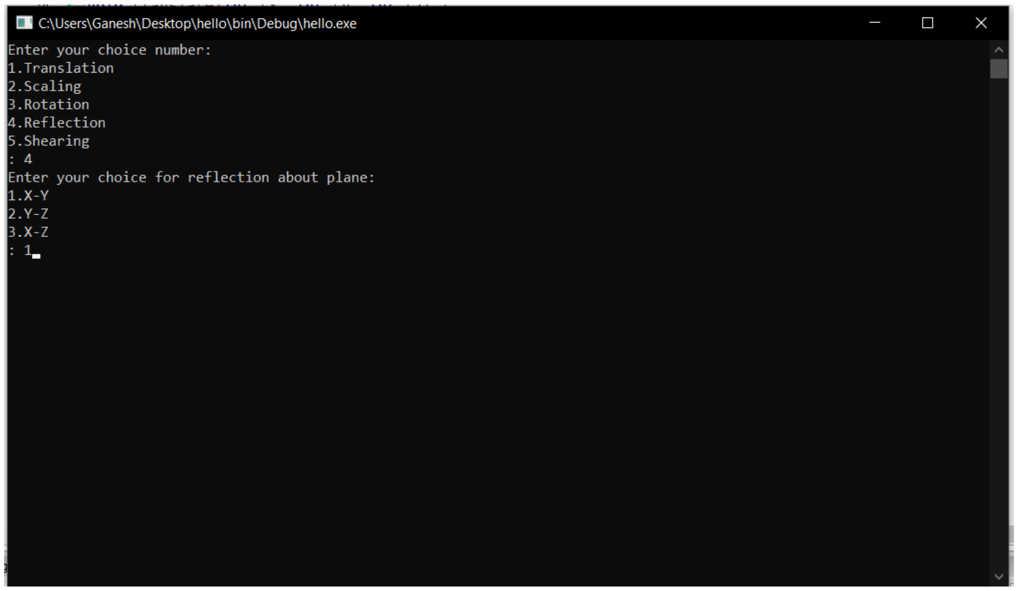
```
}
void translate(int tx, int ty, int tz)
{
    Matrix4x4 m;
    matrixSetIdentity(m);
    m[0][3] = tx;
    m[1][3] = ty;
    m[2][3] = tz;
    matrix_pre_multiply(m, theMatrix);
}
void scale(float sx, float sy, float sz)
{
    Matrix4x4 m;
    matrixSetIdentity(m);
    m[0][0] = sx;
    m[1][1] = sy;
    m[2][2] = sz;
    matrix_pre_multiply(m, theMatrix);
}
void shearx()
{
    Matrix4x4 m;
    matrixSetIdentity(m);
    m[0][1] = shy;
    m[0][2] = shz;
    matrix_pre_multiply(m, theMatrix);
}
void sheary()
{
    Matrix4x4 m;
    matrixSetIdentity(m);
    m[1][0] = shx;
    m[1][2] = shz;
    matrix_pre_multiply(m, theMatrix);
}
void shearz()
{
    Matrix4x4 m;
    matrixSetIdentity(m);
    m[2][0] = shx;
    m[2][1] = shy;
    matrix_pre_multiply(m, theMatrix);
}
void RotateX(float angle)
{
    Matrix4x4 m;
    matrixSetIdentity(m);
    angle = angle * 22 / 1260;
    m[1][1] = cos(angle);
    m[1][2] = -sin(angle);
    m[2][1] = sin(angle);
    m[2][2] = cos(angle);
    matrix_pre_multiply(m, theMatrix);
}
void RotateY(float angle)
{
    Matrix4x4 m;
    matrixSetIdentity(m);
    angle = angle * 22 / 1260;
    m[0][0] = cos(angle);
    m[0][2] = sin(angle);
    m[2][0] = -sin(angle);
    m[2][2] = cos(angle);
    matrix_pre_multiply(m, theMatrix);
}
void RotateZ(float angle)
{
    Matrix4x4 m;
    matrixSetIdentity(m);
    angle = angle * 22 / 1260;
    m[0][0] = cos(angle);
    m[0][1] = -sin(angle);
    m[1][0] = sin(angle);
    m[1][1] = cos(angle);
    matrix_pre_multiply(m, theMatrix);
}
void Reflect(void)
{
    Matrix4x4 m;
    matrixSetIdentity(m);
    switch (choiceRef)
    {
    case 1:
        m[2][2] = -1;
        break;
    case 2:
        m[0][0] = -1;
        break;
    case 3:
        m[1][1] = -1;
        break;
    }
    matrix_pre_multiply(m, theMatrix);
}
void DrawRotLine(void)
{
    switch (choiceRot)
    {
    case 1:
        glBegin(GL_LINES);
        glVertex3s(-1000, B, C);
        glVertex3s(1000, B, C);
        glEnd();
        break;
    case 2:
        glBegin(GL_LINES);
        glVertex3s(A, -1000, C);
        glVertex3s(A, 1000, C);
        glEnd();
        break;
    case 3:
        glBegin(GL_LINES);
        glVertex3s(A, B, -1000);
        glVertex3s(A, B, 1000);
        glEnd();
        break;
    case 4:
        glBegin(GL_LINES);
        glVertex3s(x1 - aa * 500, y11 - bb * 500, z1 - cc * 500);
        glVertex3s(x2 + aa * 500, y2 + bb * 500, z2 + cc * 500);
        glEnd();
        break;
    }
}
void TransformPoints(void)
{
    int i, k;
    float tmp;
    for (k = 0; k < 8; k++)
        for (i = 0; i < 3; i++)
            ptsFin[k][i] = theMatrix[i][0] * ptsIni[k][0] +
                           theMatrix[i][1] * ptsIni[k][1] +
                           theMatrix[i][2] * ptsIni[k][2] + theMatrix[i][3];
    // Realign above line while execution
}
```

```
void Axes(void)
{
    glColor3f(0.0, 0.0, 0.0); // Set the color to BLACK
    glBegin(GL_LINES);        // Plotting X-Axis
    glVertex2s(-1000, 0);
    glVertex2s(1000, 0);
    glEnd();
    glBegin(GL_LINES); // Plotting Y-Axis
    glVertex2s(0, -1000);
    glVertex2s(0, 1000);
    glEnd();
}
void Draw(float a[8][3]) // Display the Figure
{
    int i;
    glColor3f(1.0, 0.5, 1.0);
    glBegin(GL_POLYGON);
    glVertex3f(a[0][0], a[0][1], a[0][2]);
    glVertex3f(a[1][0], a[1][1], a[1][2]);
    glVertex3f(a[2][0], a[2][1], a[2][2]);
    glVertex3f(a[3][0], a[3][1], a[3][2]);
    glEnd();
    i = 0;
    glColor3f(1.0, 0.6, 0.5);
    glBegin(GL_POLYGON);
    glVertex3s(a[0 + i][0], a[0 + i][1], a[0 + i][2]);
    glVertex3s(a[1 + i][0], a[1 + i][1], a[1 + i][2]);
    glVertex3s(a[5 + i][0], a[5 + i][1], a[5 + i][2]);
    glVertex3s(a[4 + i][0], a[4 + i][1], a[4 + i][2]);
    glEnd();
    glColor3f(0.2, 0.4, 1.0);
    glBegin(GL_POLYGON);
    glVertex3f(a[0][0], a[0][1], a[0][2]);
    glVertex3f(a[3][0], a[3][1], a[3][2]);
    glVertex3f(a[7][0], a[7][1], a[7][2]);
    glVertex3f(a[4][0], a[4][1], a[4][2]);
    glEnd();
    i = 1;
    glColor3f(0.5, 0.4, 0.3);
    glBegin(GL_POLYGON);
    glVertex3s(a[0 + i][0], a[0 + i][1], a[0 + i][2]);
    glVertex3s(a[1 + i][0], a[1 + i][1], a[1 + i][2]);
    glVertex3s(a[5 + i][0], a[5 + i][1], a[5 + i][2]);
    glVertex3s(a[4 + i][0], a[4 + i][1], a[4 + i][2]);
    glEnd();
    i = 2;
    glColor3f(0.5, 0.6, 0.2);
    glBegin(GL_POLYGON);
    glVertex3s(a[0 + i][0], a[0 + i][1], a[0 + i][2]);
    glVertex3s(a[1 + i][0], a[1 + i][1], a[1 + i][2]);
    glVertex3s(a[5 + i][0], a[5 + i][1], a[5 + i][2]);
    glVertex3s(a[4 + i][0], a[4 + i][1], a[4 + i][2]);
    glEnd();
    i = 4;
    glColor3f(1.0, 0.3, 0.4);
    glBegin(GL_POLYGON);
    glVertex3f(a[0 + i][0], a[0 + i][1], a[0 + i][2]);
    glVertex3f(a[1 + i][0], a[1 + i][1], a[1 + i][2]);
    glVertex3f(a[2 + i][0], a[2 + i][1], a[2 + i][2]);
    glVertex3f(a[3 + i][0], a[3 + i][1], a[3 + i][2]);
    glEnd();
}
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    Axes();
    glColor3f(1.0, 0.0, 0.0); // Set the color to RED
    Draw(ptsIni);
    matrixSetIdentity(theMatrix);
    if (choice.find(1) != choice.end())
    {
        translate(TransDistX, TransDistY, TransDistZ);
    }
    if (choice.find(2) != choice.end())
    {
        scale(ScaleX, ScaleY, ScaleZ);
    }
    if (choice.find(3) != choice.end())
    {
        switch (choiceRot)
        {
        case 1:
            DrawRotLine();
            translate(0, -B, -C);
            RotateX(Alpha);
            translate(0, B, C);
            break;
        case 2:
            DrawRotLine();
            translate(-A, 0, -C);
            RotateY(Beta);
            translate(A, 0, C);
            break;
        case 3:
            DrawRotLine();
            translate(-A, -B, 0);
            RotateZ(Gamma);
            translate(A, B, 0);
            break;
        case 4:
            DrawRotLine();
            float MOD = sqrt((x2 - x1) * (x2 - x1) + (y2 - y11) * (y2 - y11) +
                             (z2 - z1) * (z2 - z1));
            aa = (x2 - x1) / MOD;
            bb = (y2 - y11) / MOD;
            cc = (z2 - z1) / MOD;
            translate(-x1, -y11, -z1);
            float ThetaDash;
            ThetaDash = 1260 * atan(bb / cc) / 22;
            RotateX(ThetaDash);
            RotateY(1260 * asin(-aa) / 22);
            RotateZ(Theta);
            RotateY(1260 * asin(aa) / 22);
            RotateX(-ThetaDash);
            translate(x1, y11, z1);
            break;
        }
    }
    if (choice.find(4) != choice.end())
    {
        Reflect();
    }
    if (choice.find(5) != choice.end())
    {
        if (choiceSh == 1)
        {
            shearx();
        }
        else if (choiceSh == 2)
        {
            sheary();
        }
        else
        {
            shearz();
        }
    }
    TransformPoints();
    Draw(ptsFin);
    glFlush();
}
void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    // Set the Background color to WHITE
    glOrtho(-454.0, 454.0, -250.0, 250.0, -250.0, 250.0);
    // Set the no. of Co-ordinates along X & Y axes and their gappings
    glEnable(GL_DEPTH_TEST);
    // To Render the surfaces Properly according to their depths
}
int main(int argc, char **argv)
{
    while (true)
    {
        printf("Enter your choice "
               "number:\n1.Translation\n2.Scaling\n3.Rotation\n4.Reflection\n5."
               "Shearing\n: ");
        int code;
        cin >> code;
        choice.insert(code);
        if (code == 1)
        {
            printf("Enter Translation along X, Y & Z\n: ");
            scanf("%f%f%f", &TransDistX, &TransDistY, &TransDistZ);
        }
        else if (code == 2)
        {
            printf("Enter Scaling ratios along X, Y & Z\n: ");
            scanf("%f%f%f", &ScaleX, &ScaleY, &ScaleZ);
        }
        else if (code == 3)
        {
            printf("Enter your choice for Rotation about axis:\n1.parallel to "
                   "X-axis.(y=B & z=C)\n2.parallel to Y-
axis.(x=A & z=C)\n3.parallel to Z-
axis.(x=A & y=B)\n4.Arbitrary line passing through (x1,y1,z1) & (x2,y2,z2)\n: ");
            // Realign above line while execution
            scanf("%d", &choiceRot);
            if (choiceRot == 1)
            {
                printf("Enter B & C: ");
                scanf("%f %f", &B, &C);
                printf("Enter Rot. Angle Alpha: ");
                scanf("%f", &Alpha);
            }
            else if (choiceRot == 2)
            {
                printf("Enter A & C: ");
                scanf("%f %f", &A, &C);
                printf("Enter Rot. Angle Beta: ");
                scanf("%f", &Beta);
            }
            else if (choiceRot == 3)
            {
                printf("Enter A & B: ");
                scanf("%f %f", &A, &B);
                printf("Enter Rot. Angle Gamma: ");
                scanf("%f", &Gamma);
            }
            else if (choiceRot == 4)
            {
                printf("Enter values of x1 ,y1 & z1:\n");
                scanf("%f %f %f", &x1, &y11, &z1);
                printf("Enter values of x2 ,y2 & z2:\n");
                scanf("%f %f %f", &x2, &y2, &z2);
                printf("Enter Rot. Angle Theta: ");
                scanf("%f", &Theta);
            }
            else
            {
                cout << "Invalid option opted.";
            }
        }
        else if (code == 4)
        {
            printf("Enter your choice for reflection about "
                   "plane:\n1.X-Y\n2.Y-Z\n3.X-Z\n: ");
            scanf("%d", &choiceRef);
        }
        else if (code == 5)
        {
            cout << "Enter your choice\n1.Shear X \n 2.Shear Y\n3.ShearZ\n";
            cin >> choiceSh;
            if (choiceSh == 1)
            {
                cout << "Enter shy and shz:";
                cin >> shy >> shz;
            }
            else if (choiceSh == 2)
            {
                cout << "Enter shx and shz:";
                cin >> shx >> shz;
            }
            else if (choiceSh == 3)
            {
                cout << "Enter shx and shy:";
                cin >> shx >> shy;
            }
            else
            {
                cout << "Invalid option opted";
            }
        }
        else
        {
            break;
        }
    }
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(1362, 750);
    glutInitWindowPosition(0, 0);
    glutCreateWindow(" Composite Transformations ");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

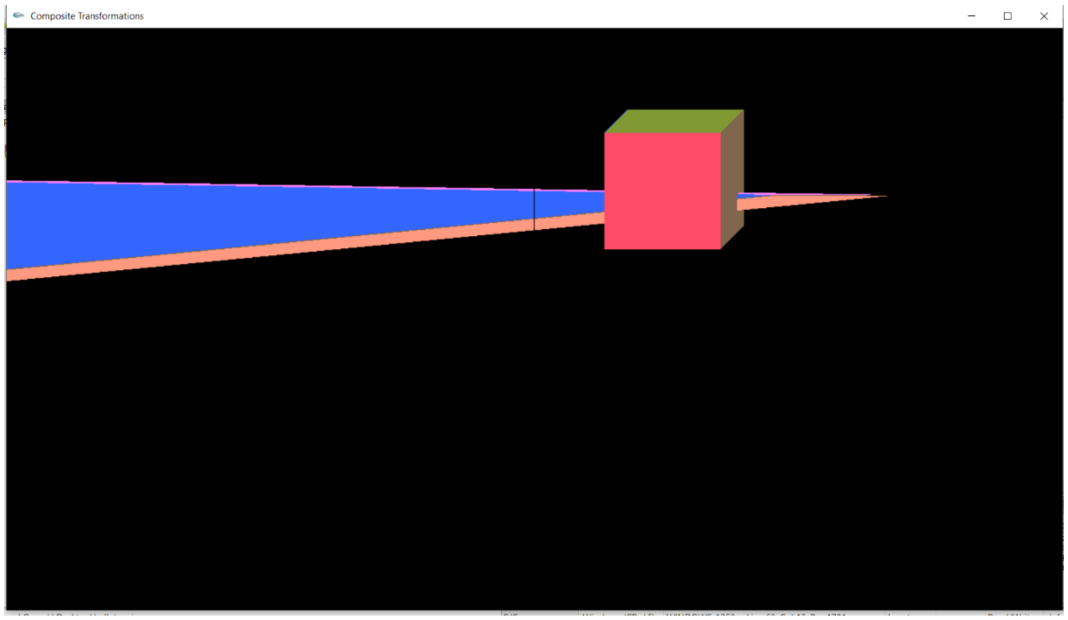
Output:-

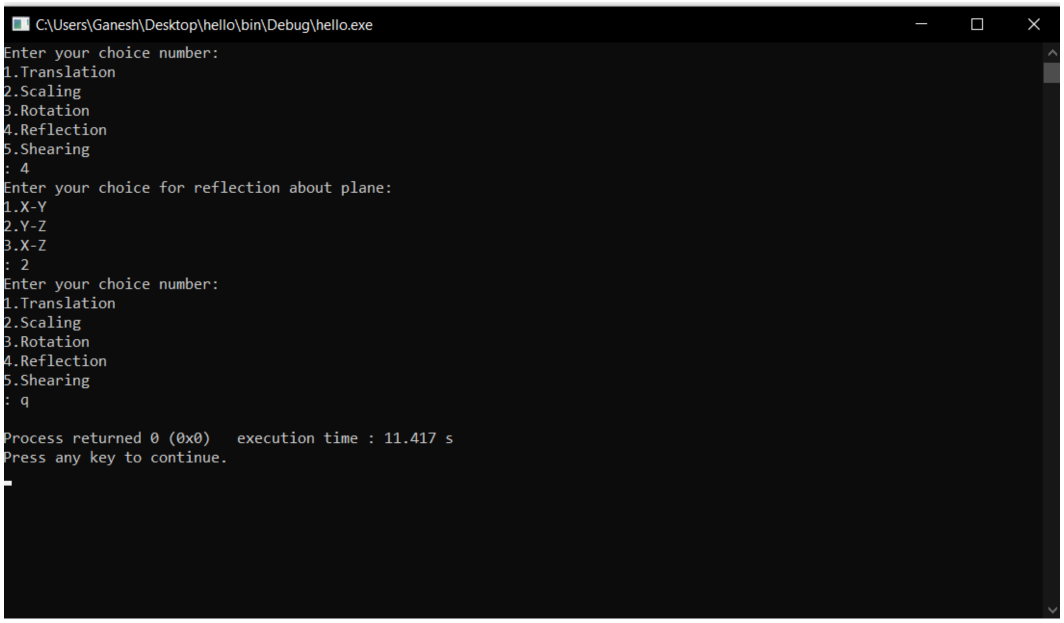


```
C:\Users\Ganesh\Desktop\hello\bin\Debug\hello.exe
Enter your choice number:
1.Translation
2.Scaling
3.Rotation
4.Reflection
5.Shearing
:
```



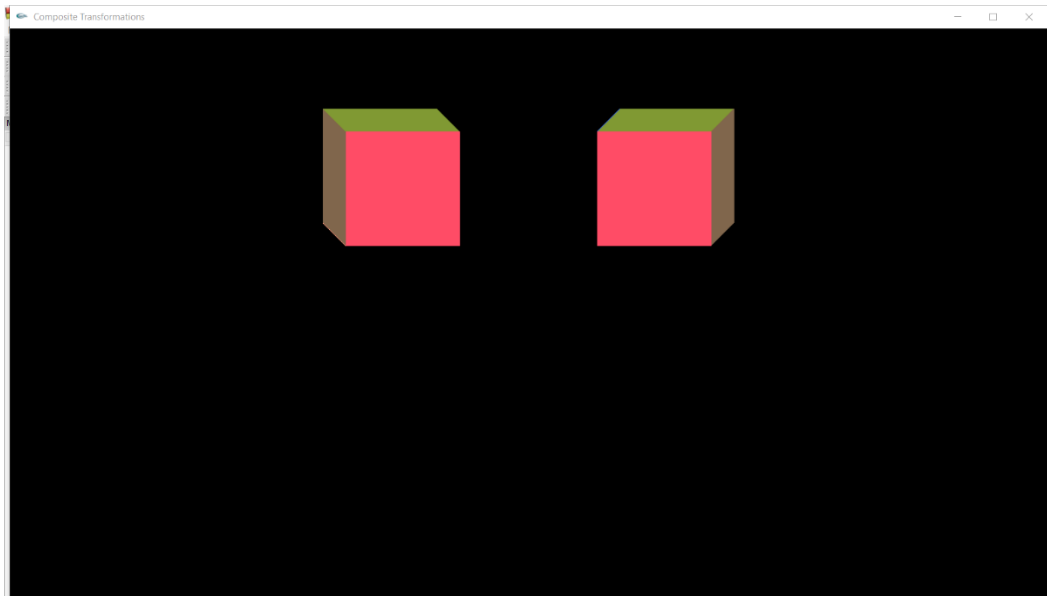
```
C:\Users\Ganesh\Desktop\hello\bin\Debug\hello.exe
Enter your choice number:
1.Translation
2.Scaling
3.Rotation
4.Reflection
5.Shearing
: 4
Enter your choice for reflection about plane:
1.X-Y
2.Y-Z
3.X-Z
: 1
```





```
Enter your choice number:
1.Translation
2.Scaling
3.Rotation
4.Reflection
5.Shearing
: 4
Enter your choice for reflection about plane:
1.X-Y
2.Y-Z
3.X-Z
: 2
Enter your choice number:
1.Translation
2.Scaling
3.Rotation
4.Reflection
5.Shearing
: q

Process returned 0 (0x0)   execution time : 11.417 s
Press any key to continue.
```


Composite Transformations