Code:

```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<graphics.h>
#include<math.h>
typedef enum
{
    TRANSLATE, SCALE, ROTATE, ROTATEAAA, SHEAR, REFLECT
}Operation;
typedef struct Matrix
{
    int rows, columns;
    double matrix[4][4];
}Matrix;
typedef struct
{
    int sides;
    Matrix* arr;
}Shape;
Shape initShape(int arr[], int sides);
Shape applyTransform(Shape shape, Operation op);
void drawShape(Shape shape);
void destroyShape(Shape shape);
Matrix initMatrix(int rows, int columns);
void printMatrix(Matrix m);
Matrix multiplyMatrix(Matrix m1, Matrix m2);
Matrix setMatDataAt(int data, Matrix mat, int row, int column);
Matrix initHomoTransform();
Matrix translate(int tx, int ty);
Matrix scale(double sx, double sy);
Matrix rotate(int angle);
Matrix rotateAbout(int angle, int anchorx, int anchory);
Matrix shear(double shx, int x_axis);

Shape initShape(int arr[], int sides)
{
    int i, j, k;
    Shape shape;
    shape.sides = sides;
    shape.arr = (Matrix*)malloc((sides + 1)* sizeof(Matrix));
    k = 0;
    for( i = 0; i < sides + 1 ; i++)
    {
        *(shape.arr + i) = initMatrix(3, 1);
        *(shape.arr + i) = setMatDataAt(1, *(shape.arr + i), 2, 0);
        for( j = 0; j < 2; j++)
        {
            *(shape.arr + i) = setMatDataAt(arr[k], *(shape.arr + i), j, 0 );
            k++;
        }
    }
    return shape;
}
Shape applyTransform(Shape shape, Operation op)
{
    int i;
    double d1, d2, d3, d4;
    Matrix transform = initHomoTransform();
    switch(op)
    {
        case TRANSLATE:
            printf("Enter tx & ty: ");
            scanf("%lf%lf", &d1, &d2);
            transform = translate((int) d1, (int) d2);
            break;
        case SCALE:
            printf("Enter sx & asy: ");
            scanf("%lf%lf", &d1, &d2);
            transform = translate(- (*(shape.arr)).matrix[0][0],
                                  - (*(shape.arr)).matrix[1][0]);
            transform = multiplyMatrix(scale(d1,d2), transform);
            transform = multiplyMatrix(
                translate((*(shape.arr)).matrix[0][0], (*(shape.arr)).matrix[1][0])
                , transform);
            break;
        case ROTATE:
            printf("Enter the angle to rotate with: ");
            scanf("%lf", &d1);
            transform = translate(- (*(shape.arr)).matrix[0][0],
                                  - (*(shape.arr)).matrix[1][0]);
            transform = multiplyMatrix(rotate((int)d1), transform);
            transform = multiplyMatrix(
                translate((*(shape.arr)).matrix[0][0], (*(shape.arr)).matrix[1][0])
                , transform);
            break;
        case ROTATEAAA:
            printf("Enter the point about which to rotate: ");
            scanf("%lf%lf", &d1, &d2);
            printf("Enter the angle: ");
            scanf("%lf", &d3);
            transform = translate(- (int)d1, - (int)d2);
            transform = multiplyMatrix(rotate((int)d3), transform);
            transform = multiplyMatrix(
                translate((int) d1, (int) d2)
                , transform);
            break;
        case SHEAR:
            printf("Enter the shear factor: ");
            scanf("%lf", &d1);
            printf("Enter 1 for shearing about X-axis or 0 for shearing about Y-axis: ");
            scanf("%d", &i);
            transform = translate(- (*(shape.arr)).matrix[0][0],
                                  -(*(shape.arr)).matrix[1][0]);
            transform = multiplyMatrix(shear(d1, i), transform);
            transform = multiplyMatrix(
                translate((*(shape.arr)).matrix[0][0], (*(shape.arr)).matrix[1][0])
                , transform);
                break;
        case REFLECT:
            for(i = 0; i < shape.sides + 1; i++)
            {
                (shape.arr + i)->matrix[0][0] = ((double) getmaxx() - (shape.arr + i)->matrix[0][0]);
            }
            return shape;
            break;
        default:
        printf("Shouldn't Be Here");
    }
    for( i = 0; i < shape.sides + 1; i++)
    {
       *(shape.arr + i) = multiplyMatrix(transform, *(shape.arr + i));
    }
    return shape;
}

void drawShape(Shape shape)
{
    int i, j, k;
    int result2[20];
    k = 0;
    for( i = 0; i < shape.sides + 1 ; i++)
    {
        for( j = 0; j < 2; j++)
        {
```

```c
            // printMatrix(*(shape.arr + i));
            result2[k] = (*(shape.arr + i)).matrix[j][0];
            k++;
        }
    }
    drawpoly(shape.sides + 1, result2);
    getch();
    cleardevice();
}
void destroyShape(Shape shape)
{
    int i;
    for( i = 0; i < shape.sides + 1; i++)
    {
       free((shape.arr + i));
    }
}
Matrix initMatrix(int rows, int columns)
{
    int i,j;
    Matrix m;
    m.rows = rows;
    m.columns = columns;
    for(i = 0; i < rows; i++)
    {
        for(j = 0; j < columns; j++)
        {
            m.matrix[i][j] = 0;
        }
    }
    return m;
}
void printMatrix(Matrix m)
{
    int i,j;
    for(i = 0; i < m.rows; i++)
    {
        for(j = 0; j < m.columns; j++)
        {
            printf("%lf", m.matrix[i][j]);
        }
        printf("\n");
    }
    printf("\n");

}
Matrix multiplyMatrix(Matrix m1, Matrix m2)
```

```c
{
    int i, j, k;
    Matrix mult = initMatrix(m1.rows, m2.columns);
    for( i = 0; i < m1.rows; i++)
    {
        for( j = 0; j < m2.columns; j++)
        {
            for( k = 0; k < m2.rows; k++)
            {
                mult.matrix[i][j] = mult.matrix[i][j] + m1.matrix[i][k] * m2.matrix[k][j];
            }
        }
    }
    return mult;
}
Matrix setMatDataAt(int data, Matrix mat, int row, int column)
{
    mat.matrix[row][column] = data;
    return mat;
}
Matrix initHomoTransform()
{
    int i,j;
    Matrix m;
    m.rows = 3;
    m.columns = 3;
    for(i = 0; i < m.rows; i++)
    {
        for(j = 0; j < m.columns; j++)
        {
            if( i == j )
            {
                m.matrix[i][j] = 1;
            }
            else
            {
                m.matrix[i][j] = 0;
            }
        }
    }
    return m;
}
Matrix translate(int tx, int ty)
{
    Matrix m;
    m = initHomoTransform();
    m.matrix[0][2] = tx;
    m.matrix[1][2] = ty;
    return m;
}
Matrix scale(double sx, double sy)
{
    Matrix m;
    m = initHomoTransform();
    m.matrix[0][0] = sx;
    m.matrix[1][1] = sy;
    return m;
}
Matrix shear(double sh, int x_axis)
{
    Matrix m = initHomoTransform();
    if(x_axis)
    {
        m.matrix[0][1] = sh;
    }
    else
    {
        m.matrix[1][0] = sh;
    }
    return m;
}
Matrix rotate(int angle)
{
    Matrix m;
    double inRadians = (angle * 3.1415926535 / 180);
    m = initHomoTransform();
    m.matrix[0][0] = cos(inRadians);
    m.matrix[0][1] = -sin(inRadians);
    m.matrix[1][0] = sin(inRadians);
    m.matrix[1][1] = cos(inRadians);
    return m;
}

int main()
{
    int gd = DETECT, gm, length, sides, i, j, k;
    int vertices[20], result2[20];
    Shape shapeT, shapeS, shapeR, shapeSh, shapeRe;
    Matrix transform = initHomoTransform();
    vertices[0] = 200;
    vertices[1] = 250;
    vertices[2] = 250;
    vertices[3] = 250;
    vertices[4] = 250;
    vertices[5] = 300;
    vertices[6] = 200;
    vertices[7] = 300;
    vertices[8] = 200;
    vertices[9] = 250;
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
    printf("Enter the number of sides: ");
    scanf("%d", &sides);
    shapeT = initShape(vertices, sides);
    shapeS = initShape(vertices, sides);
    shapeSh = initShape(vertices, sides);
    shapeR = initShape(vertices, sides);
    shapeRe = initShape(vertices, sides);
    drawShape(shapeT);
    shapeT = applyTransform(shapeT, TRANSLATE);
    drawShape(shapeT);
    shapeS = applyTransform(shapeS, SCALE);
    drawShape(shapeS);
    shapeR = applyTransform(shapeR, ROTATE);
    drawShape(shapeR);
    shapeSh = applyTransform(shapeSh, SHEAR);
    drawShape(shapeSh);
    printf("Reflection: ");
    shapeRe = applyTransform(shapeSh, REFLECT);
    drawShape(shapeRe);
    getch();
    closegraph();
    return 0;
}
```