

Code:

```
1  #include<stdio.h>
2  #include<conio.h>
3  #include<stdlib.h>
4  #include<graphics.h>
5  #include<math.h>
6  typedef enum
7  {
8      TRANSLATE, SCALE, ROTATE, ROTATEAAA, SHEAR, REFLECT
9  }Operation;
10 typedef struct Matrix
11 {
12     int rows, columns;
13     double matrix[4][4];
14 }Matrix;
15 typedef struct
16 {
17     int sides;
18     Matrix* arr;
19 }Shape;
20 Shape initShape(int arr[], int sides);
21 Shape applyTransform(Shape shape, Operation op);
22 void drawShape(Shape shape);
23 void destroyShape(Shape shape);
24 Matrix initMatrix(int rows, int columns);
25 void printMatrix(Matrix m);
26 Matrix multiplyMatrix(Matrix m1, Matrix m2);
27 Matrix setMatDataAt(int data, Matrix mat, int row, int column);
28 Matrix initHomoTransform();
29 Matrix translate(int tx, int ty);
30 Matrix scale(double sx, double sy);
31 Matrix rotate(int angle);
32 Matrix rotateAbout(int angle, int anchorx, int anchory);
33 Matrix shear(double shx, int x_axis);
34
35 Shape initShape(int arr[], int sides)
36 {
37     int i, j, k;
38     Shape shape;
39     shape.sides = sides;
40     shape.arr = (Matrix*)malloc((sides + 1)* sizeof(Matrix));
41     k = 0;
42     for( i = 0; i < sides + 1 ; i++)
43     {
44         *(shape.arr + i) = initMatrix(3, 1);
45         *(shape.arr + i) = setMatDataAt(1, *(shape.arr + i), 2, 0);
```

```

46     for( j = 0; j < 2; j++)
47     {
48         *(shape.arr + i) = setMatDataAt(arr[k], *(shape.arr + i), j, 0 );
49         k++;
50     }
51 }
52 return shape;
53 }
54 Shape applyTransform(Shape shape, Operation op)
55 {
56     int i;
57     double d1, d2, d3, d4;
58     Matrix transform = initHomoTransform();
59     switch(op)
60     {
61         case TRANSLATE:
62             printf("Enter tx & ty: ");
63             scanf("%lf%lf", &d1, &d2);
64             transform = translate((int) d1, (int) d2);
65             break;
66         case SCALE:
67             printf("Enter sx & asy: ");
68             scanf("%lf%lf", &d1, &d2);
69             transform = translate(- (*(shape.arr)).matrix[0][0],
70                                 - (*(shape.arr)).matrix[1][0]);
71             transform = multiplyMatrix(scale(d1,d2), transform);
72             transform = multiplyMatrix(
73                 translate((*(shape.arr)).matrix[0][0],
74 ↪ (*(shape.arr)).matrix[1][0])
75                 , transform);
76             break;
77         case ROTATE:
78             printf("Enter the angle to rotate with: ");
79             scanf("%lf", &d1);
80             transform = translate(- (*(shape.arr)).matrix[0][0],
81                                 - (*(shape.arr)).matrix[1][0]);
82             transform = multiplyMatrix(rotate((int)d1), transform);
83             transform = multiplyMatrix(
84 ↪ translate((*(shape.arr)).matrix[0][0],
85             (*(shape.arr)).matrix[1][0])
86             , transform);
87             break;
88         case ROTATEAAA:
89             printf("Enter the point about which to rotate: ");
90             scanf("%lf%lf", &d1, &d2);
91             printf("Enter the angle: ");
92             scanf("%lf", &d3);

```

```

91         transform = translate(- (int)d1, - (int)d2);
92         transform = multiplyMatrix(rotate((int)d3), transform);
93         transform = multiplyMatrix(
94             translate((int) d1, (int) d2)
95             , transform);
96         break;
97     case SHEAR:
98         printf("Enter the shear factor: ");
99         scanf("%lf", &d1);
100        printf("Enter 1 for shearing about X-axis or 0 for shearing about
↪ Y-axis: ");
101        scanf("%d", &i);
102        transform = translate(- (*(shape.arr)).matrix[0][0],
103                               -(*(shape.arr)).matrix[1][0]);
104        transform = multiplyMatrix(shear(d1, i), transform);
105        transform = multiplyMatrix(
106            translate(*(shape.arr).matrix[0][0],
↪ (*(shape.arr)).matrix[1][0])
107            , transform);
108        break;
109    case REFLECT:
110        for(i = 0; i < shape.sides + 1; i++)
111        {
112            (shape.arr + i)->matrix[0][0] = ((double) getmaxx() -
↪ (shape.arr + i)->matrix[0][0]);
113        }
114        return shape;
115        break;
116    default:
117        printf("Shouldn't Be Here");
118    }
119    for( i = 0; i < shape.sides + 1; i++)
120    {
121        *(shape.arr + i) = multiplyMatrix(transform, *(shape.arr + i));
122    }
123    return shape;
124 }
125
126 void drawShape(Shape shape)
127 {
128     int i, j, k;
129     int result2[20];
130     k = 0;
131     for( i = 0; i < shape.sides + 1 ; i++)
132     {
133         for( j = 0; j < 2; j++)
134         {

```

```

135         // printMatrix(*(shape.arr + i));
136         result2[k] = (*(shape.arr + i)).matrix[j][0];
137         k++;
138     }
139 }
140 drawpoly(shape.sides + 1, result2);
141 getch();
142 cleardevice();
143 }
144 void destroyShape(Shape shape)
145 {
146     int i;
147     for( i = 0; i < shape.sides + 1; i++)
148     {
149         free((shape.arr + i));
150     }
151 }
152 Matrix initMatrix(int rows, int columns)
153 {
154     int i,j;
155     Matrix m;
156     m.rows = rows;
157     m.columns = columns;
158     for(i = 0; i < rows; i++)
159     {
160         for(j = 0; j < columns; j++)
161         {
162             m.matrix[i][j] = 0;
163         }
164     }
165     return m;
166 }
167 void printMatrix(Matrix m)
168 {
169     int i,j;
170     for(i = 0; i < m.rows; i++)
171     {
172         for(j = 0; j < m.columns; j++)
173         {
174             printf("%1f", m.matrix[i][j]);
175         }
176         printf("\n");
177     }
178     printf("\n");
179 }
180 }
181 Matrix multiplyMatrix(Matrix m1, Matrix m2)

```

```

182 {
183     int i, j, k;
184     Matrix mult = initMatrix(m1.rows, m2.columns);
185     for( i = 0; i < m1.rows; i++)
186     {
187         for( j = 0; j < m2.columns; j++)
188         {
189             for( k = 0; k < m2.rows; k++)
190             {
191                 mult.matrix[i][j] = mult.matrix[i][j] + m1.matrix[i][k] *
↪ m2.matrix[k][j];
192             }
193         }
194     }
195     return mult;
196 }
197 Matrix setMatDataAt(int data, Matrix mat, int row, int column)
198 {
199     mat.matrix[row][column] = data;
200     return mat;
201 }
202 Matrix initHomoTransform()
203 {
204     int i,j;
205     Matrix m;
206     m.rows = 3;
207     m.columns = 3;
208     for(i = 0; i < m.rows; i++)
209     {
210         for(j = 0; j < m.columns; j++)
211         {
212             if( i == j )
213             {
214                 m.matrix[i][j] = 1;
215             }
216             else
217             {
218                 m.matrix[i][j] = 0;
219             }
220         }
221     }
222     return m;
223 }
224 Matrix translate(int tx, int ty)
225 {
226     Matrix m;
227     m = initHomoTransform();

```

```

228     m.matrix[0][2] = tx;
229     m.matrix[1][2] = ty;
230     return m;
231 }
232 Matrix scale(double sx, double sy)
233 {
234     Matrix m;
235     m = initHomoTransform();
236     m.matrix[0][0] = sx;
237     m.matrix[1][1] = sy;
238     return m;
239 }
240 Matrix shear(double sh, int x_axis)
241 {
242     Matrix m = initHomoTransform();
243     if(x_axis)
244     {
245         m.matrix[0][1] = sh;
246     }
247     else
248     {
249         m.matrix[1][0] = sh;
250     }
251     return m;
252 }
253 Matrix rotate(int angle)
254 {
255     Matrix m;
256     double inRadians = (angle * 3.1415926535 / 180);
257     m = initHomoTransform();
258     m.matrix[0][0] = cos(inRadians);
259     m.matrix[0][1] = - sin(inRadians);
260     m.matrix[1][0] = sin(inRadians);
261     m.matrix[1][1] = cos(inRadians);
262     return m;
263 }
264
265 int main()
266 {
267     int gd = DETECT, gm, length, sides, i, j, k;
268     int vertices[20], result2[20];
269     Shape shapeT, shapeS, shapeR, shapeSh, shapeRe;
270     Matrix transform = initHomoTransform();
271     vertices[0] = 200;
272     vertices[1] = 250;
273     vertices[2] = 250;
274     vertices[3] = 250;

```

```

275     vertices[4] = 250;
276     vertices[5] = 300;
277     vertices[6] = 200;
278     vertices[7] = 300;
279     vertices[8] = 200;
280     vertices[9] = 250;
281     initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
282     printf("Enter the number of sides: ");
283     scanf("%d", &sides);
284     shapeT = initShape(vertices, sides);
285     shapeS = initShape(vertices, sides);
286     shapeSh = initShape(vertices, sides);
287     shapeR = initShape(vertices, sides);
288     shapeRe = initShape(vertices, sides);
289     drawShape(shapeT);
290     shapeT = applyTransform(shapeT, TRANSLATE);
291     drawShape(shapeT);
292     shapeS = applyTransform(shapeS, SCALE);
293     drawShape(shapeS);
294     shapeR = applyTransform(shapeR, ROTATE);
295     drawShape(shapeR);
296     shapeSh = applyTransform(shapeSh, SHEAR);
297     drawShape(shapeSh);
298     printf("Reflection: ");
299     shapeRe = applyTransform(shapeSh, REFLECT);
300     drawShape(shapeRe);
301     getch();
302     closegraph();
303     return 0;
304 }

```